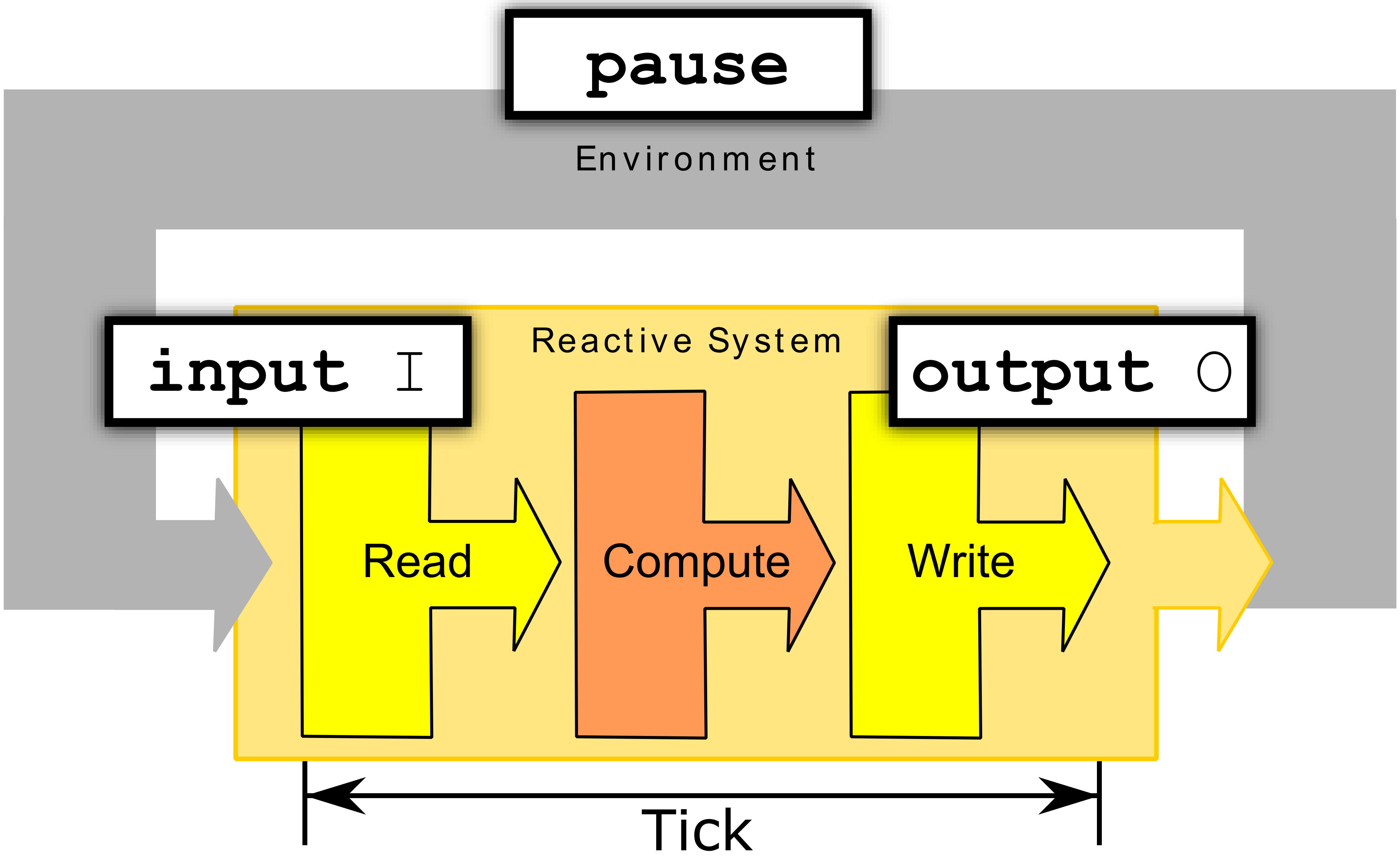
Sequential Constructiveness, SCL and SCCharts

Incorporating synchrony in conventional languages

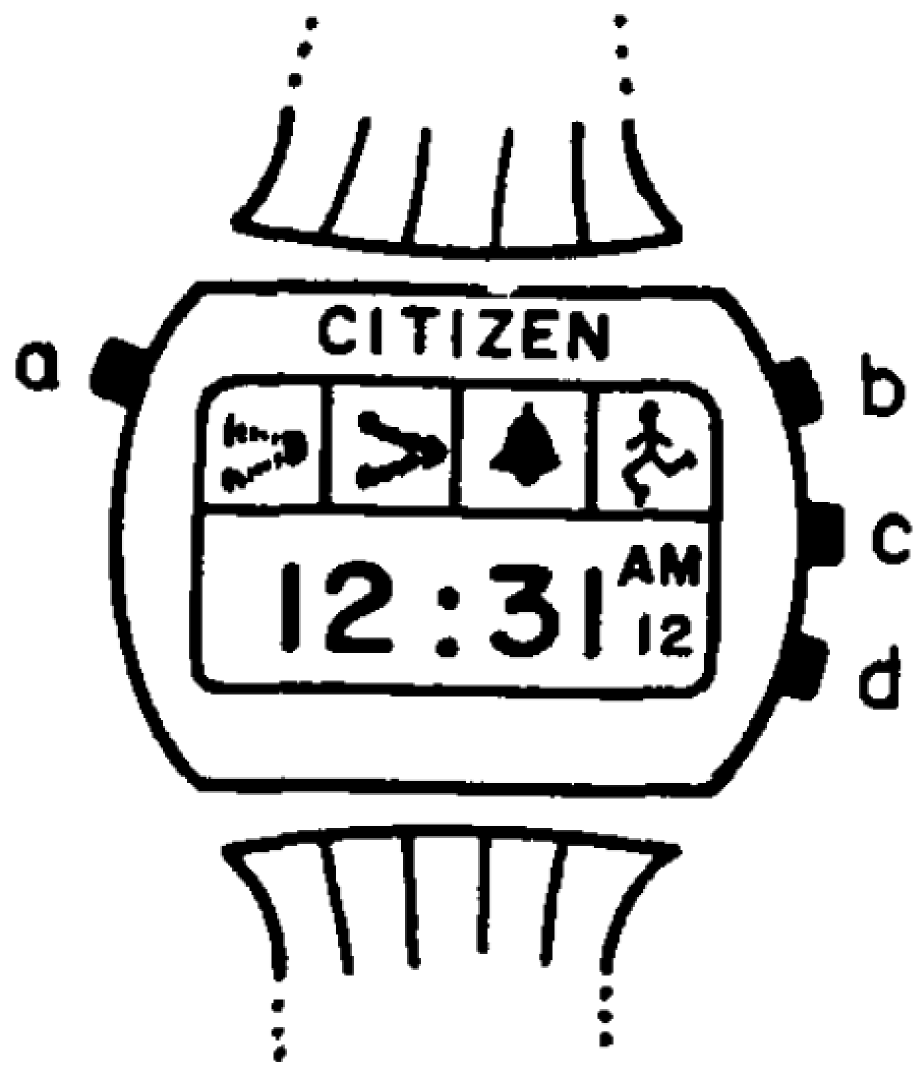
Reinhard von Hanxleden (U Kiel)

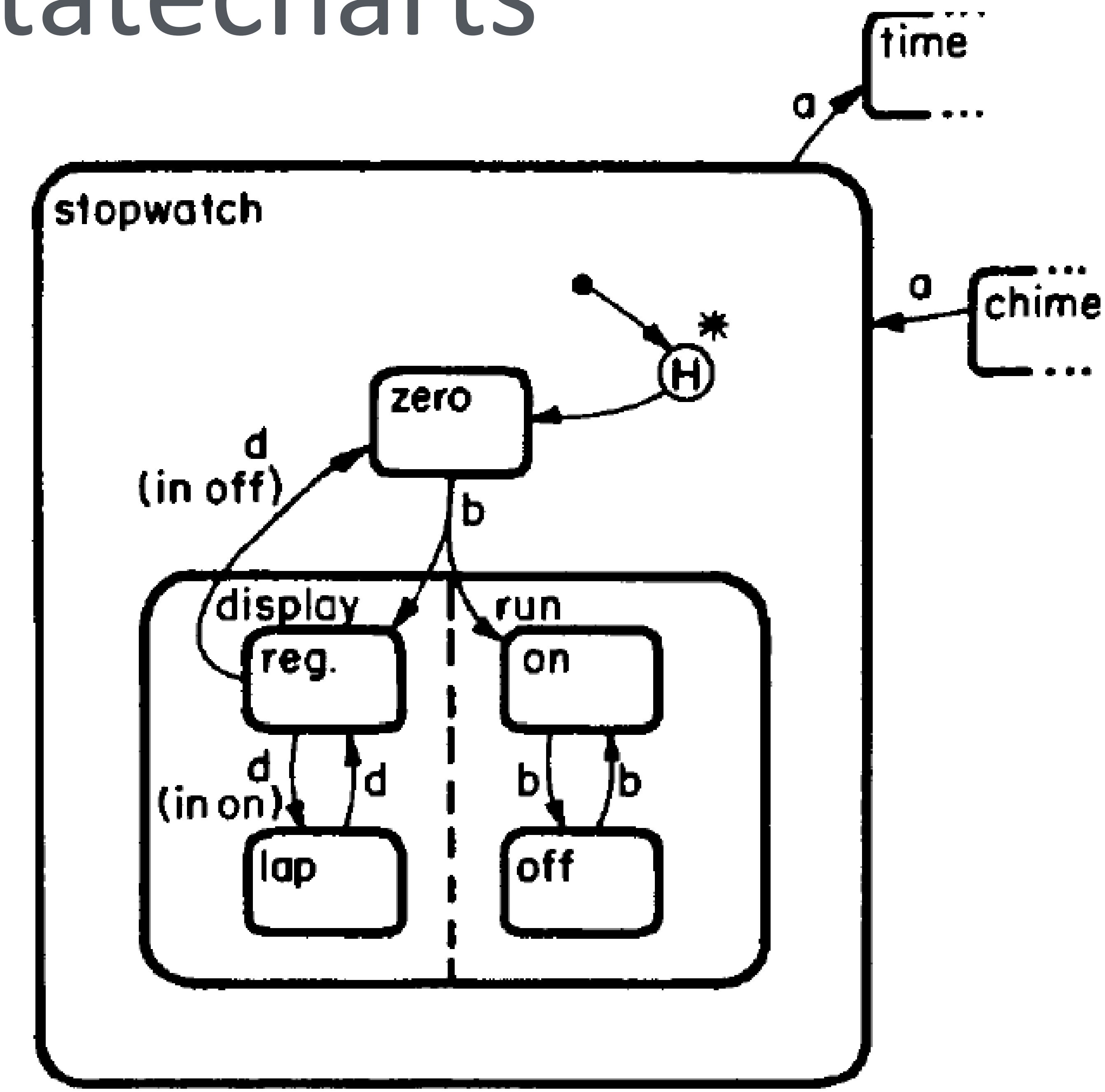
14 March 2018, Collège de France

Reactive Systems



1980s: Statecharts





[David Harel,
Statecharts: A Visual Formalism for Complex Systems,
Science of Computer Programming, 1987]

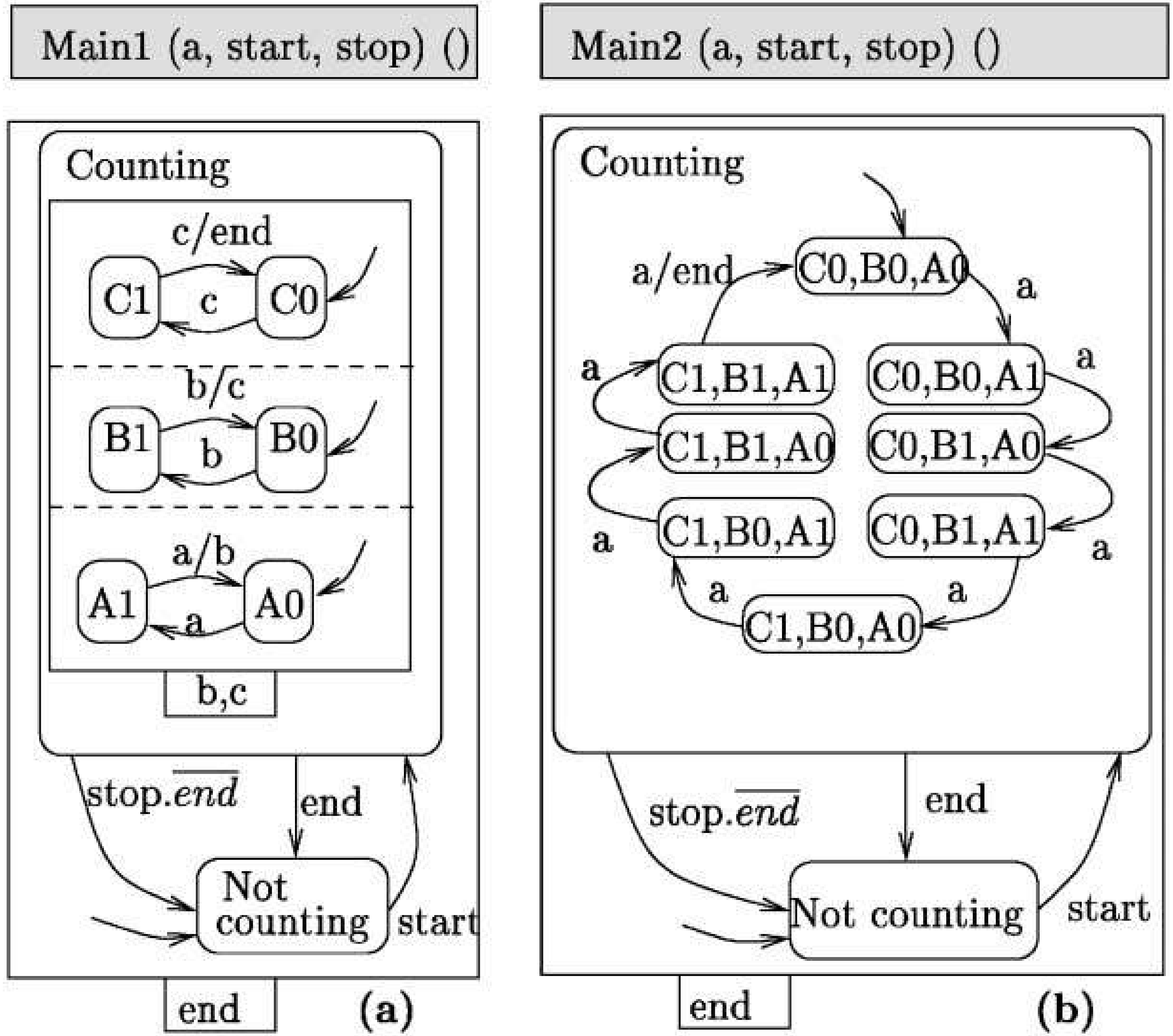
1990s: Many Statecharts

	Variant									
	1	2	3	4	5	6	7	8	9	10
graphical / textual	g	g	g	g	g	g	g	g/t	g/t	g
negated trigger event	+	+	+	+	+	+	+	+	+	-
timeout event	+	-	-	-	-	-	-	+	+	+[9]
timed transition	-	-	-	-	-	-	-	-	-	-
disjunction of trigger events	-	+	+	+	+	-[13]	-[13]	+	+	-

Statecharts Feature	Variant Number																				
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
graphical / textual	g	g	g	g	g	g	g	g/t	g/t	g	g	g/t	g	g	g	g	g	g	g	g	g
negated trigger event	+	+	+	+	+	+	+	+	+	-	+	-	+	+	-	-	-	-	+	+	+
timeout event	+	-	-	-	-	-	-	+	+	+[9]	-	-	-	+	+[4]	+[4]	+[4]	+[4]	+[4]	+[4]	+[4]
timed transition	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	+	+	+	+[2]	+[2]	+
disjunction of trigger events	-	+	+	+	+	-[13]	-[13]	+	+	-	-	-	+	+	-	-	-	-	-	-	+
trigger condition	+	+	-	-	-	-[13]	-[13]	+[3]	-	+	-	-	-	+	-	-	-	-	+	+	-
state reference	+	+	-	-	-	-[13]	-[13]	+[3]	-	+	-	-	-	+	-	-	-	-	-	-	-
assignment to variable	+	+	-	-	-	-[13]	-[13]	-	-	+	-	-	-	+	-	-	-	-	+	+	-
inter-level transition	+	+	-[14]	-[14]	-[14]	+	+	+	+	+	-	-	+	+	+	+	+	+	-	-	-
history mechanism	+	+	-	-	-	-	-	-	-	-	-	-	+	-	-	-	-	-	-	-	-
operational/denotatio.	-	o	o	o	o	o	[11]	d	d	o	o	d	o	o	d	d	d	d	o	o	?
compositional	-	-	-	-	-	-	-	+	+	-	+	+		-	+	+	+	+	+	+	+
synchrony hypothesis	+	+	+	+	+	+	+	+	+	+[7]	+	+	+	-	-	-	-	-	-[8]	-[8]	-
deterministic	-	-	-	-	-	-	-	-	-	-	+	-	-	-	-	-	-	-	-	-	-
interleav./true concurr.	i	i	i	i	i	i	i	i	i	i	i	i	i	i	t	t	t	t	[15]	[15]	?
discrete/contin. time	d	d	d	d	d	d	d	d	d	d	d	d	d	d	c	c	c	c	c	c	c
globally consistent	-	-	-	+	+	+	+	-[13]	+	+[5]	-	+[5]	-	-	+[5]	+[5]	+[5]	+[5]	-	-	-
causal	+	+	+	+	-	+	+	+	+	+[1]	-[12]	+	+	+	+	+	+	+	+	+	+
instantaneous state	?	-	-	-	-	-	-	-	-	+	-	+	-	-	-	-	+	+	+	+	+
finite transition no.	?	+	+	+	+	+	+	+	+	-	+	-	+	+	+	+	+	+	-	-	+
priorities	-	-	-[14]	-[14]	-[14]	-[10]	-[10]	-	-	-	+	-	-	+	-	+	-	+	-	-	+
non-preempt. interrupt	?	-	-[14]	-[14]	-[14]	-	-	-	-	?	+	-	-	-	-	-	+	+	?	?	+
preemptive interrupt	?	+	-[14]	-[14]	-[14]	+	+	+	+	+	+	-	+	+	+	+	-	-	+	+	+
distinc. int./ext. event	+	+	+	+	+	+	+	+	+	+[6]	+	+	+	-	+	+	+	+	+	+	+
local event	-	-	-	-	-	-	-	+	+	+	+	+	-	-	-	-	-	-	-	-	+
discrete/contin. event	d	d	d	d	d	d	d	d	d	d	d	d	d	d	c	c	c	c	d	d	d

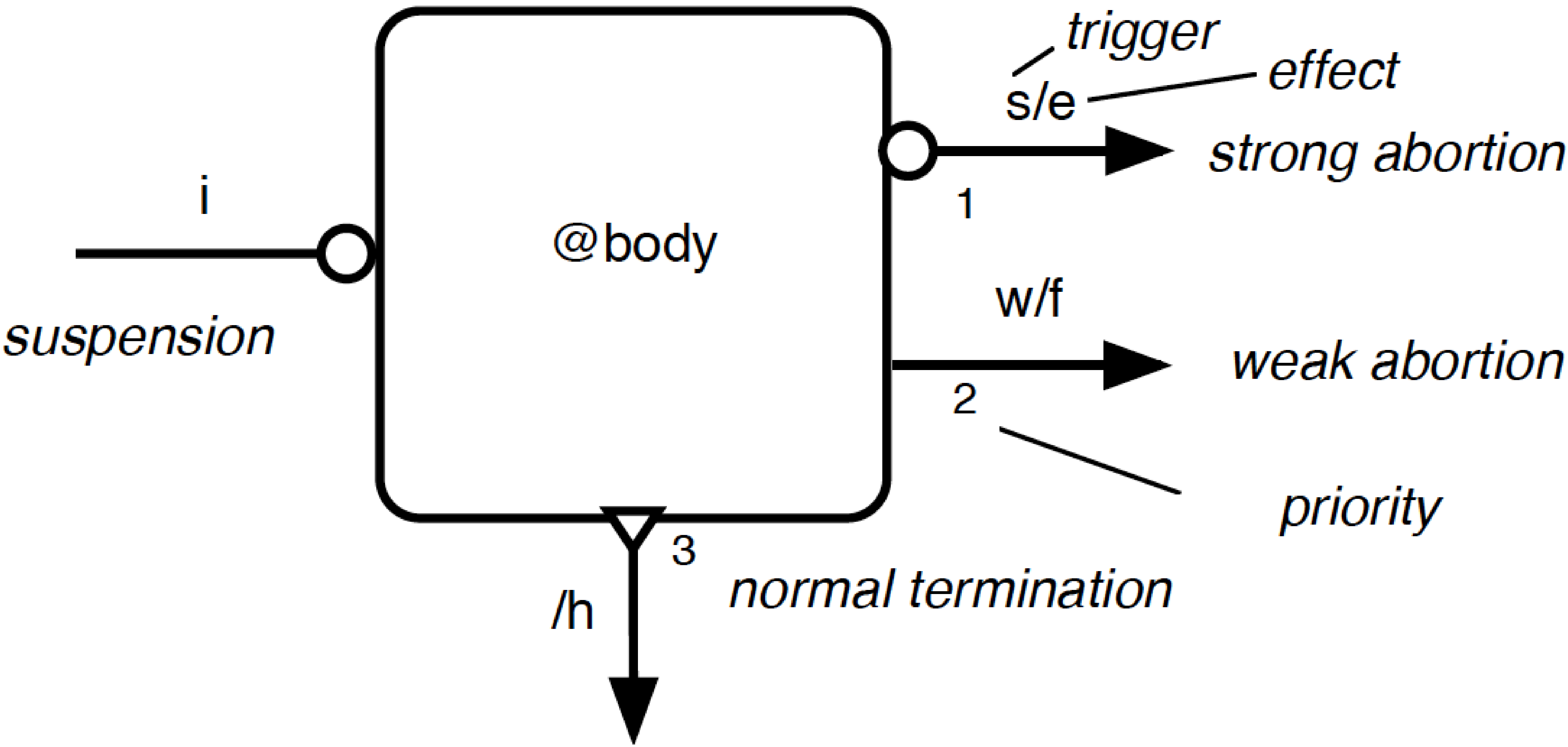
[Michael von der Beeck,
A Comparison of Statecharts Variants,
Formal Techniques in Real-Time and Fault-Tolerant Systems, LNCS 863, 1994]

1991: Argos



[Florence Maraninchi,
The Argos language: Graphical Representation of Automata and Description of Reactive Systems, IEEE Workshop on Visual Languages, Kobe, Japan, 1991]

1995: SyncCharts, a.k.a. Safe State Machines



[Charles André,
SyncCharts: A Visual Representation of Reactive Behaviors,
Research Report 95-52, I3S, Sophia Antipolis, 1995]

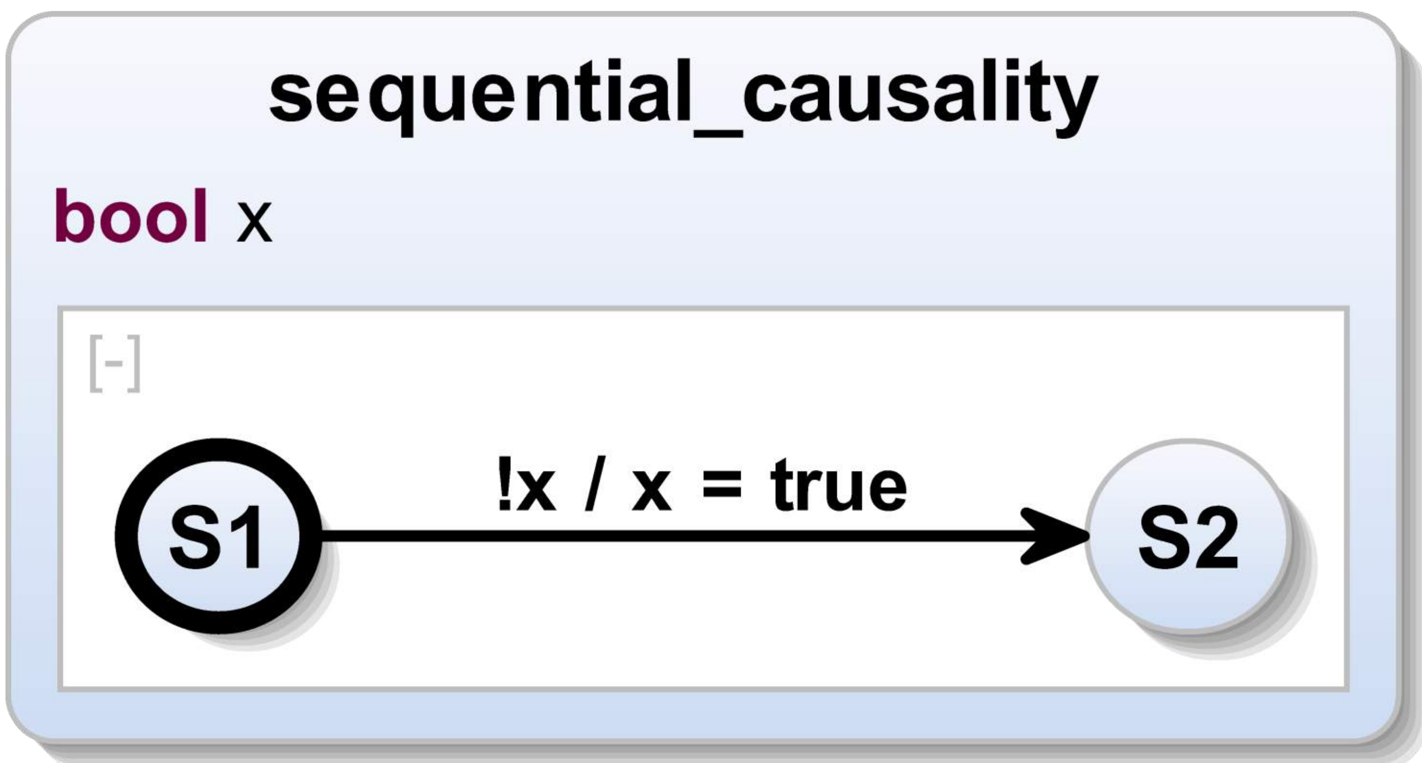
SCCharts – Motivation

Limitations of Strict Synchrony

```
if (!x) {
  x = true;
}
```

Good C

Limitations of Strict Synchrony



Bad SyncChart

Good SCChart

Limitations of Strict Synchrony

```
present x else
  emit x
end
```

Bad Esterel

Good SCEst

SCCharts – Motivation

Preserve nice properties of synchronous programming

- Determinacy
- Sound semantic basis
- Efficient synthesis

Reduce the pain

- Make it easy to adapt for mainstream programmer
- Reject only models where determinacy is compromised

Model of Computation

Sequential Constructiveness

Sequential control flow overrides „write before read“

Writes visible only to reads that are

1. sequential successors or
2. concurrent

[v. Hanxleden, Mendler, et al.,
Sequentially Constructive Concurrency—A Conservative Extension of the Synchronous Model of Computation,
ACM TECS ’14]

SCCharts – MoC

It's all about scheduling variable accesses within reaction …

- Sequential accesses to x: unconstrained
- Concurrent accesses to x („iur protocol"):
  init ⇒ updates ⇒ reads
  `x = 1 …` `x += 2 … x += 5 …` `y = x … z = x`
- Concurrent accesses may lead to causality cycles – compiler must reject those

SCCharts – MoC

SCChart / SCEst program is …

… **SC**, „is sequentially constructive,“ if

1. there exist runs obeying iur protocol
2. all such runs produce same result

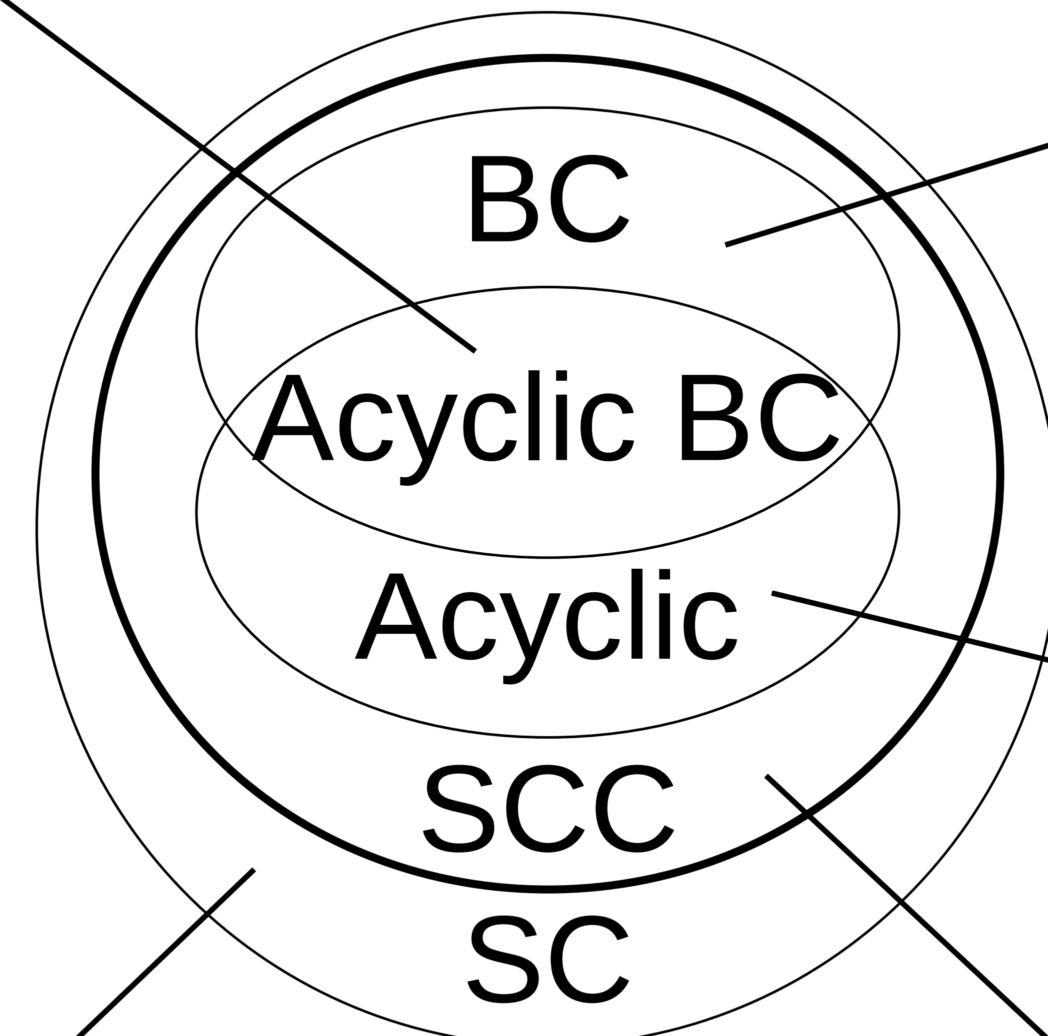
… **SCC**, „corresponds to sequentially constructive circuit,“ if it is SC and does not „speculate“

Program Classes

```
module XYpresent:
[
  emit X
||
  present X then emit Y end
]
```

```
module XYelse:
[
  present X then emit Y end
||
  present Y else emit X end
]
```

```
module XY:
[
  present X then emit Y end
||
  present Y then emit X end
]
```




```
module ABBA:
[
  present A and I
    then emit B end
||
  present B and not I
    then emit A end
||
  present A or B
    then emit O end
]
```

```
module OffOn:
present S then emit T end;
emit S;
present S then emit U end
```

```
module Dynamic:
loop
  emit S;
  present I then pause end;
  present S then emit T end;
  present I else pause end
end
```

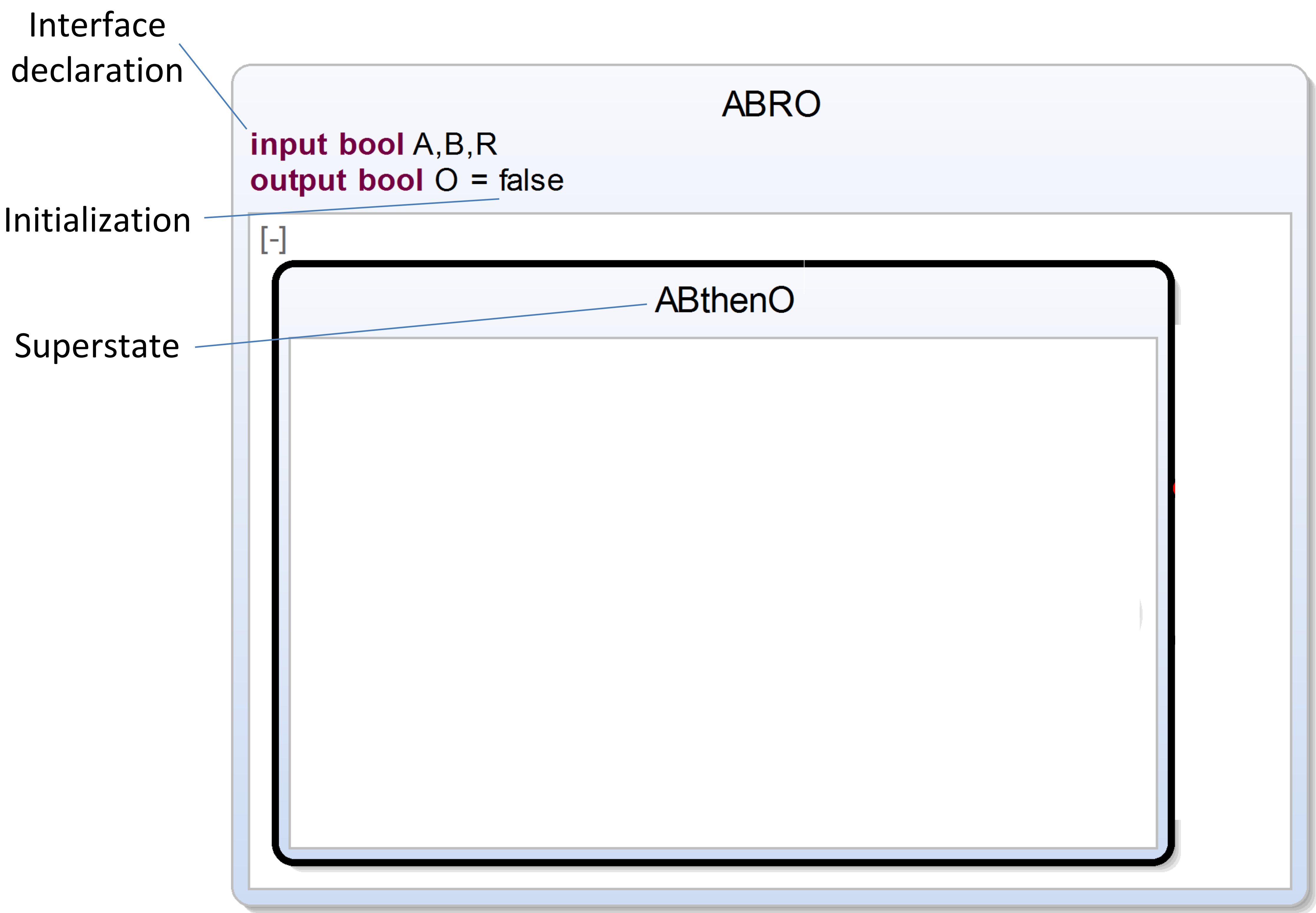
SCCharts – The Language(s)

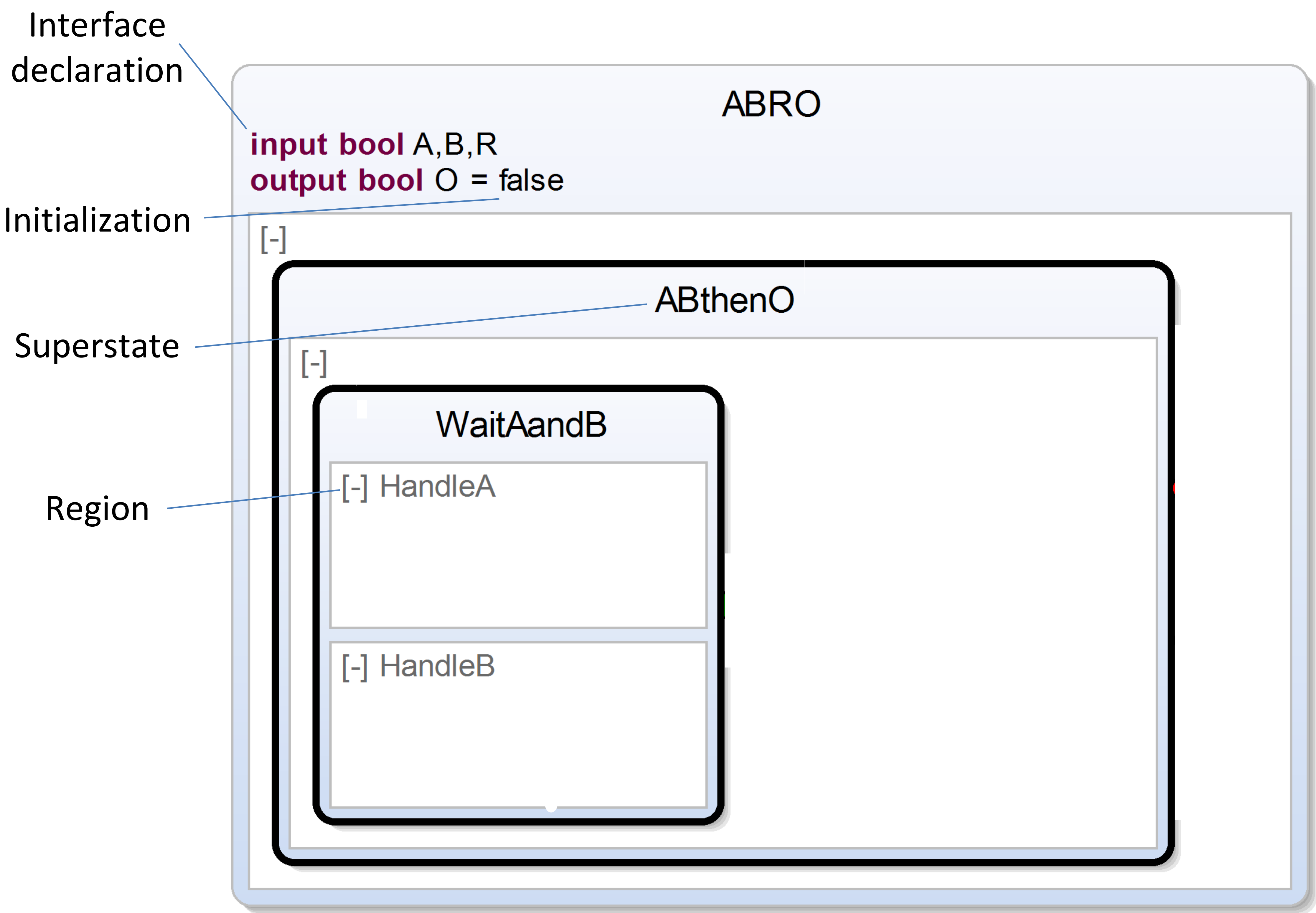
Interface declaration

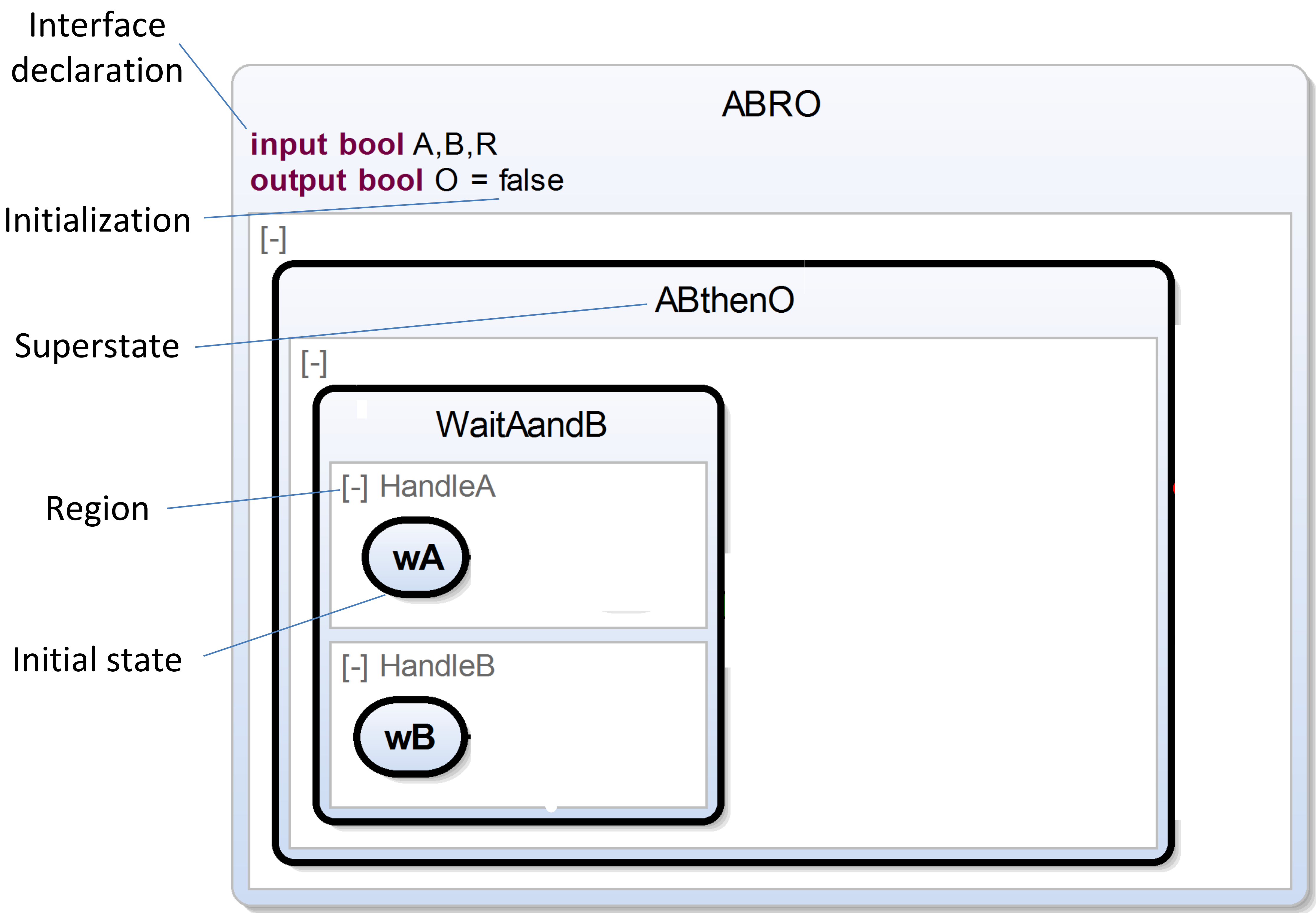
ABRO

input bool A,B,R
output bool O = false

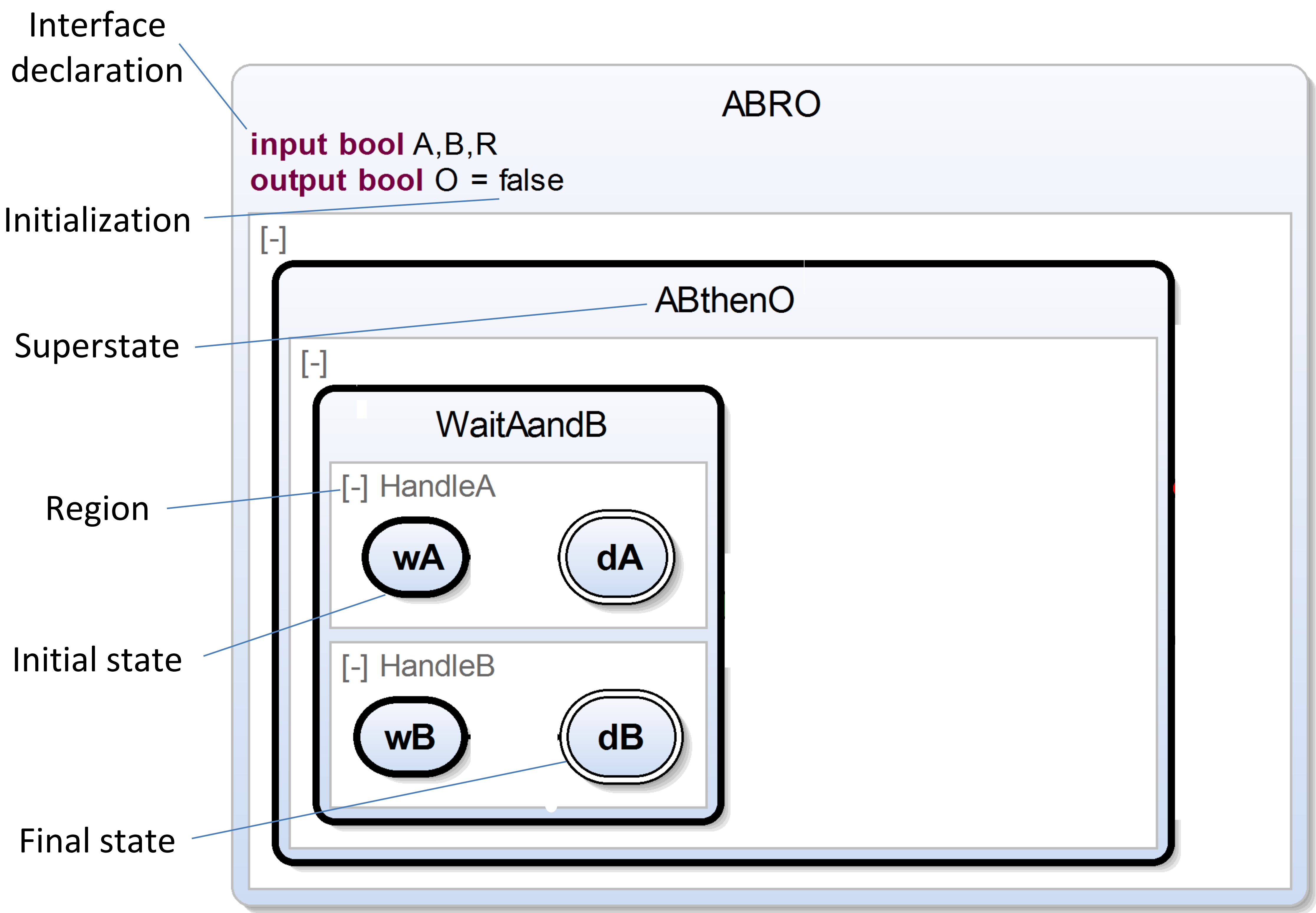
Initialization


Interface declaration
Initialization
Superstate
ABRO
input bool A,B,R
output bool O = false
[-]
ABthenO

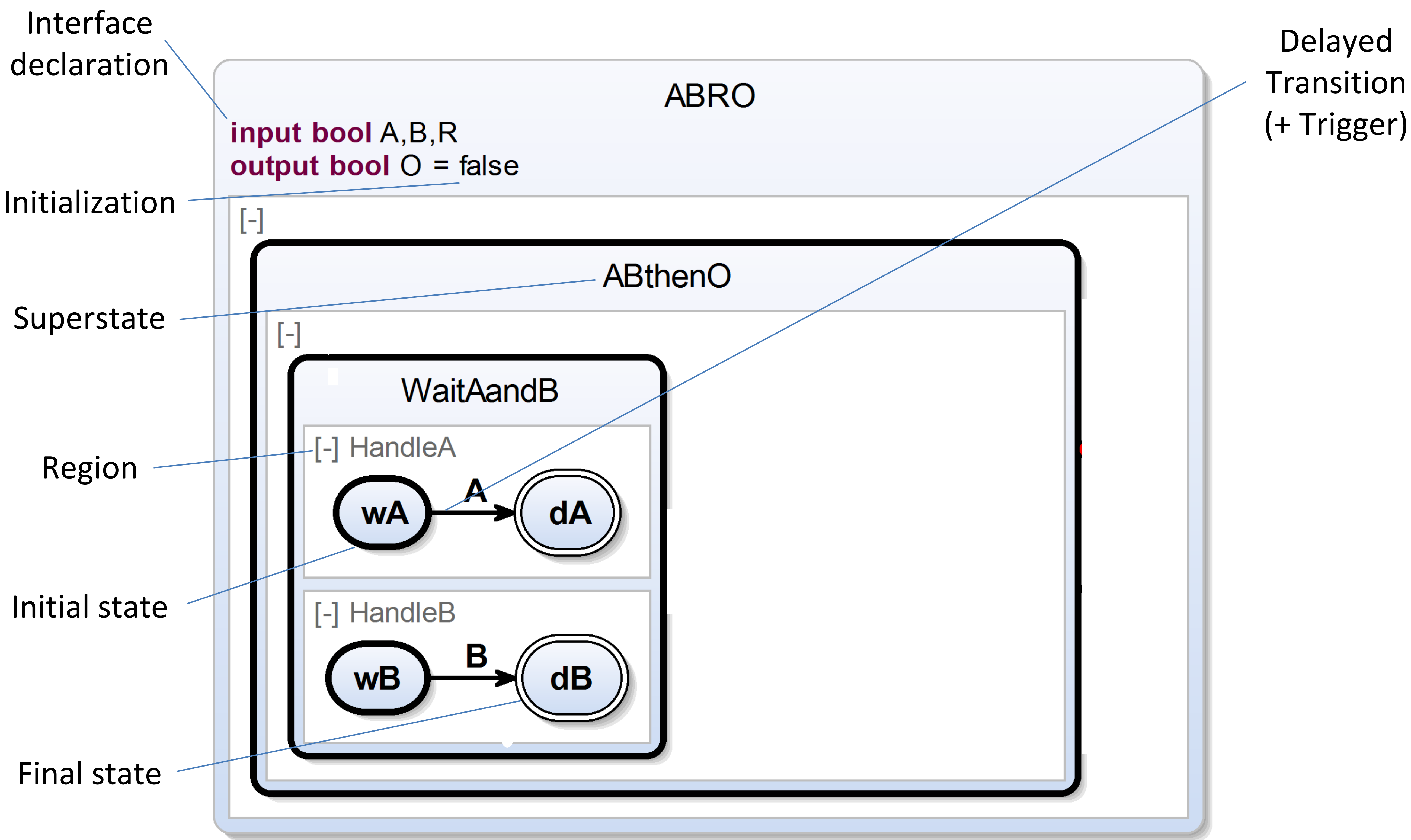

Interface declaration
Initialization
Superstate
Region
ABRO
input bool A,B,R
output bool O = false
[-]
ABthenO
[-]
WaitAandB
[-] HandleA
[-] HandleB

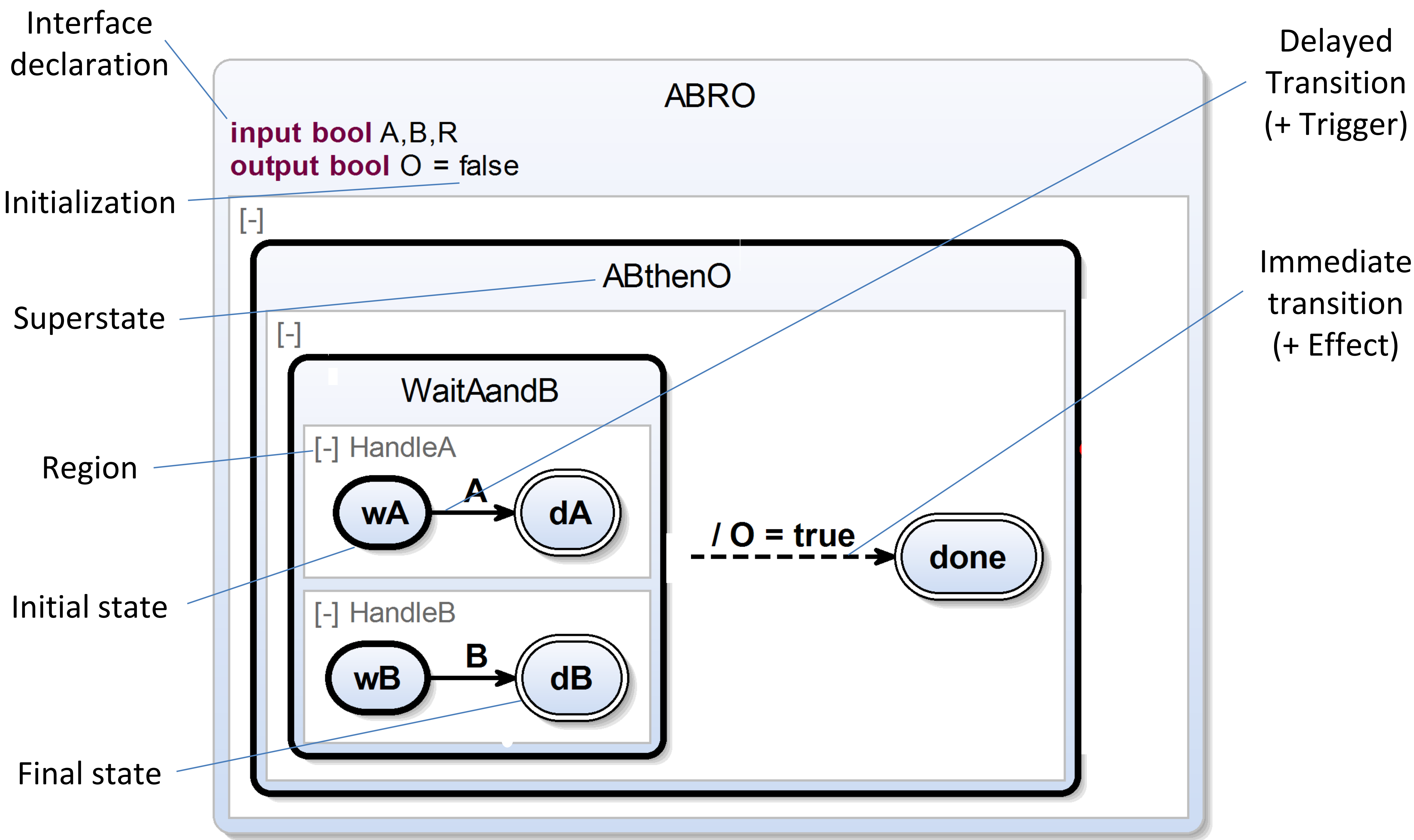


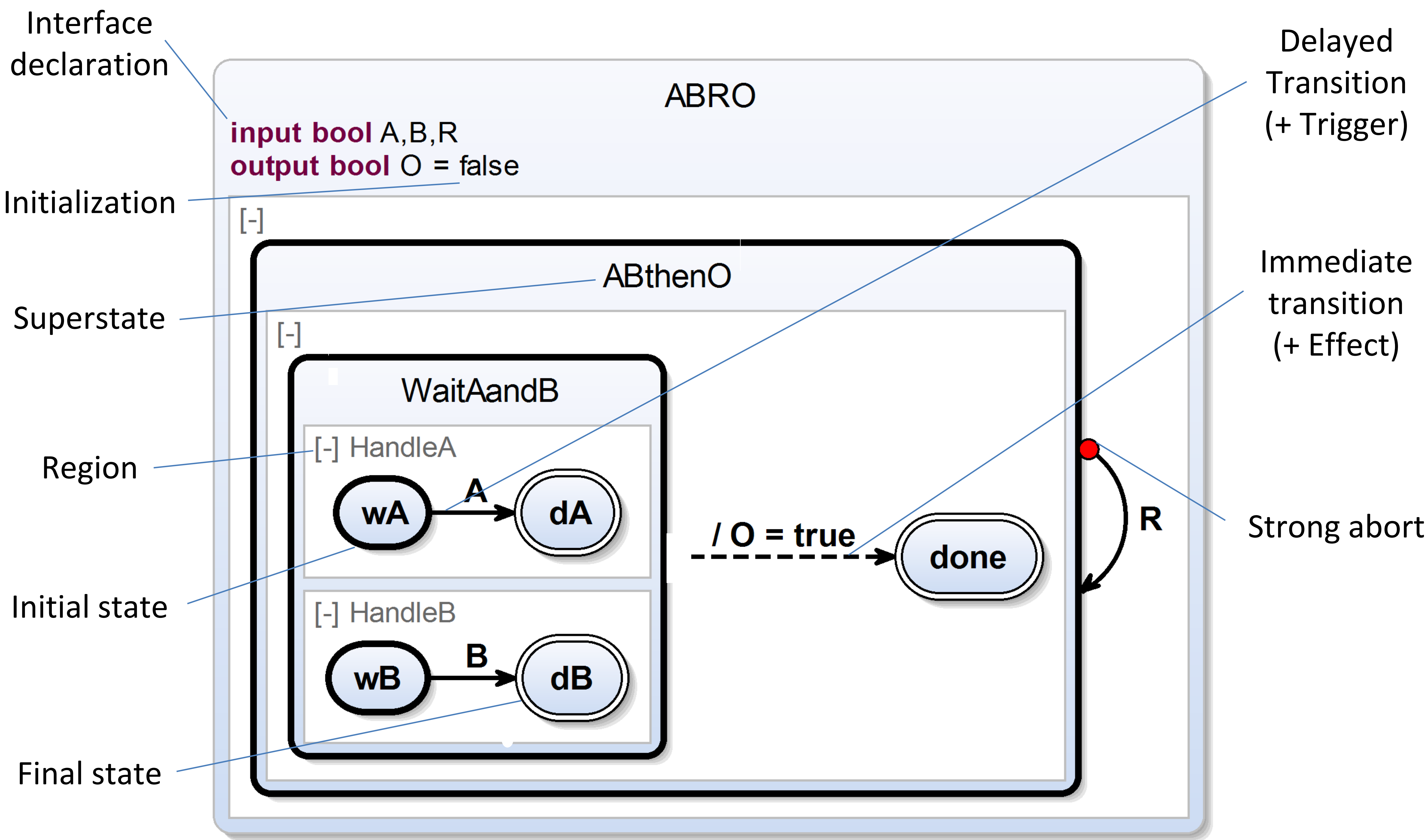
Interface declaration
Initialization
Superstate
Region
Initial state
ABRO
input bool A,B,R
output bool O = false
[-]
ABthenO
[-]
WaitAandB
[-] HandleA
wA
[-] HandleB
wB

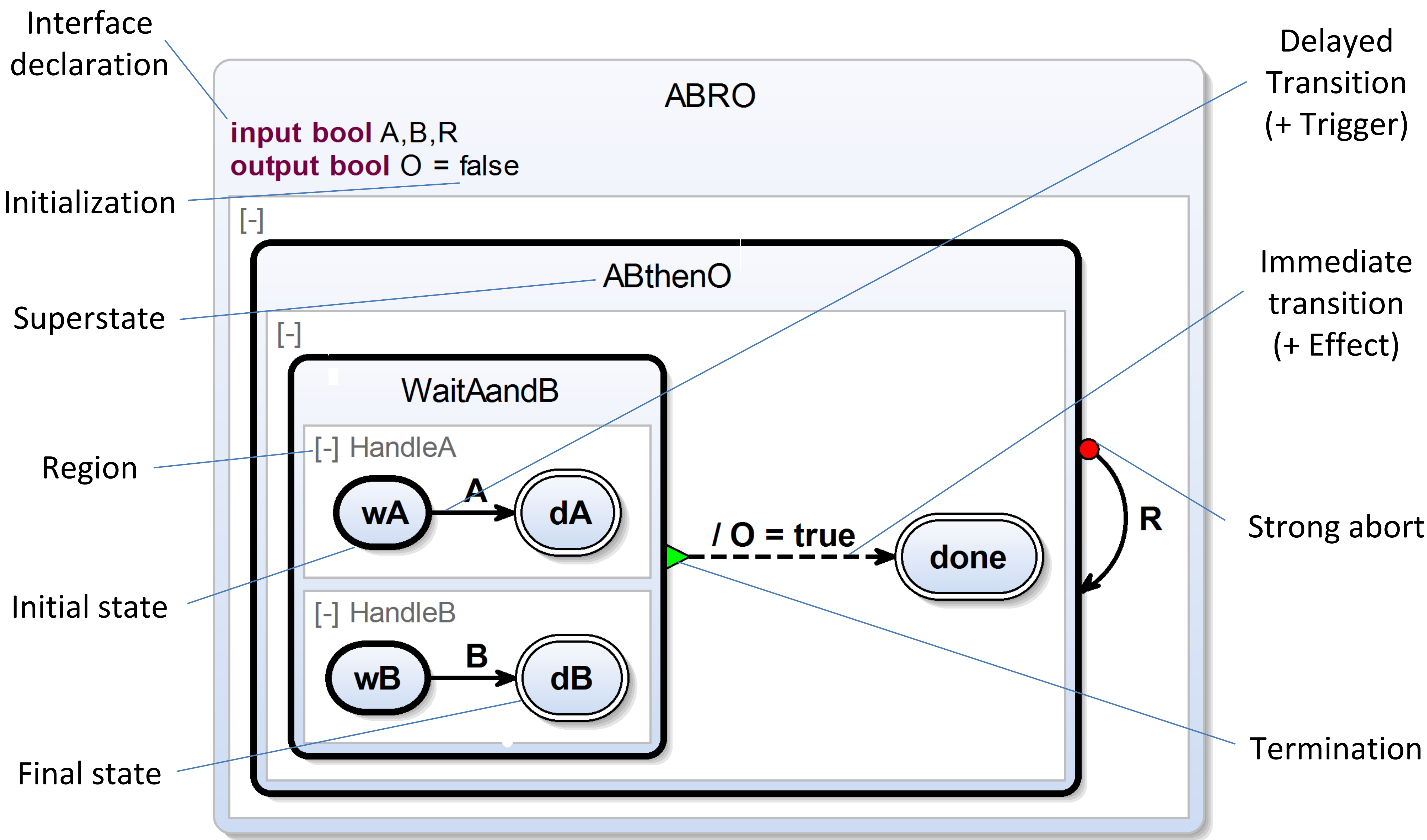


Interface declaration
Initialization
Superstate
Region
Initial state
Final state
ABRO
input bool A,B,R
output bool O = false
[-]
ABthenO
[-]
WaitAandB
[-] HandleA
wA
dA
[-] HandleB
wB
dB

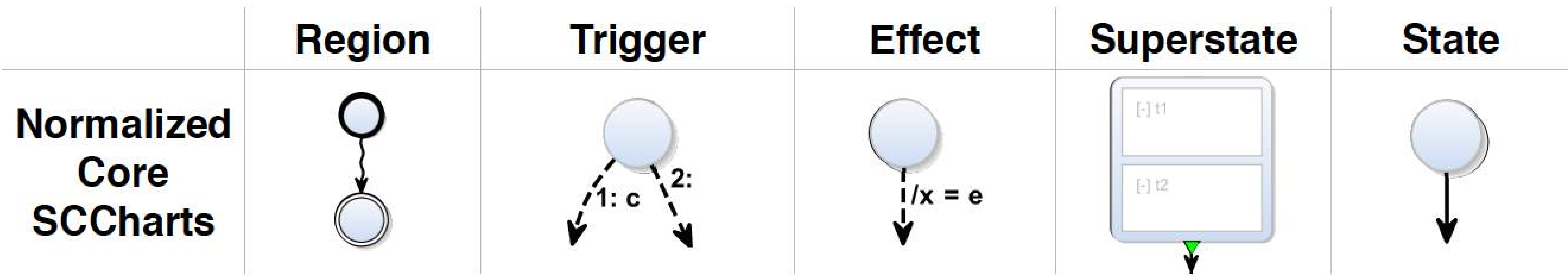

Interface declaration
Initialization
Superstate
Region
Initial state
Final state
Delayed Transition (+ Trigger)
ABRO
input bool A,B,R
output bool O = false
[-]
ABthenO
[-]
WaitAandB
[-] HandleA
wA
A
dA
[-] HandleB
wB
B
dB


Interface declaration
Initialization
Superstate
Region
Initial state
Final state
Delayed Transition (+ Trigger)
Immediate transition (+ Effect)
ABRO
input bool A,B,R
output bool O = false
[-]
ABthenO
[-]
WaitAandB
[-] HandleA
wA
A
dA
[-] HandleB
wB
B
dB
/ O = true
done

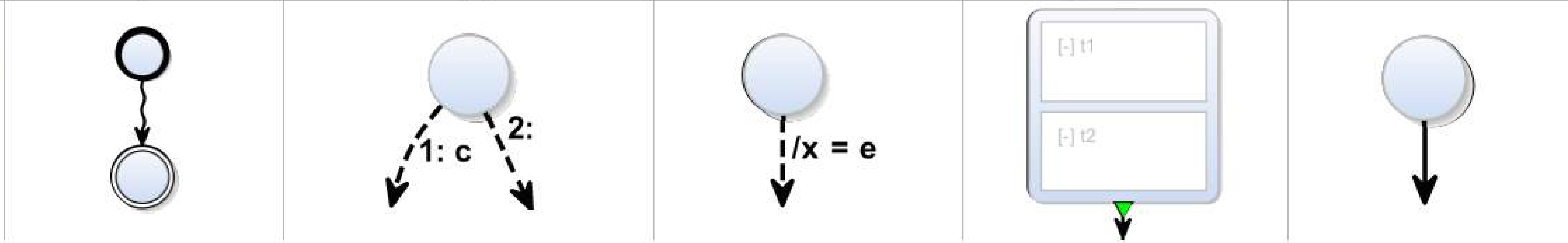

Interface declaration
Initialization
Superstate
Region
Initial state
Final state
ABRO
input bool A,B,R
output bool O = false
[-]
ABthenO
[-]
WaitAandB
[-] HandleA
wA
A
dA
[-] HandleB
wB
B
dB
/ O = true
done
R
Delayed Transition (+ Trigger)
Immediate transition (+ Effect)
Strong abort


Interface declaration
Initialization
Superstate
Region
Initial state
Final state
ABRO
input bool A,B,R
output bool O = false
[-]
ABthenO
[-]
WaitAandB
[-] HandleA
wA
A
dA
[-] HandleB
wB
B
dB
/ O = true
done
R
Delayed Transition (+ Trigger)
Immediate transition (+ Effect)
Strong abort
Termination

SCChart Building Blocks

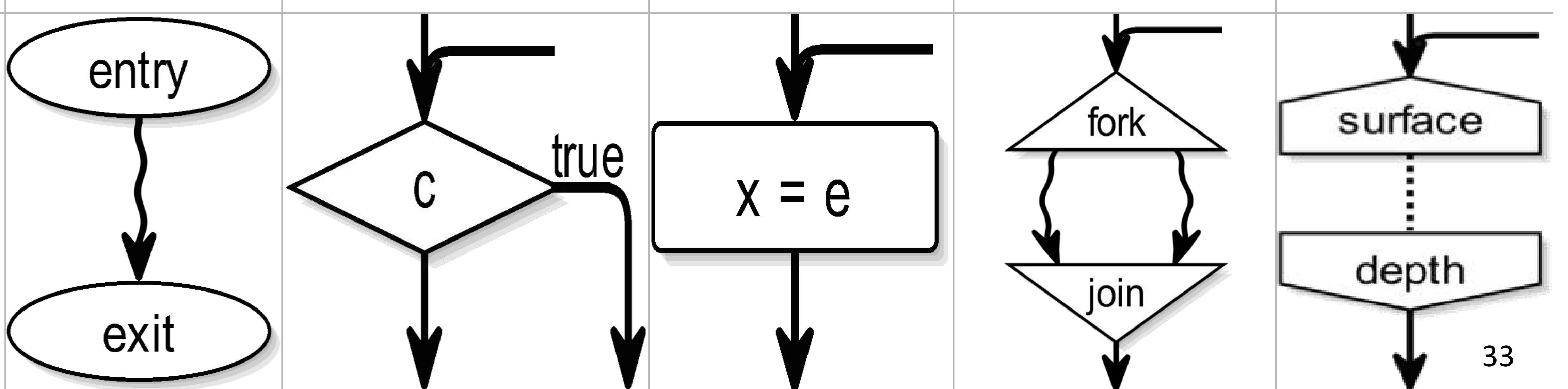


	Region	Trigger	Effect	Superstate	State
Normalized Core SCCharts					

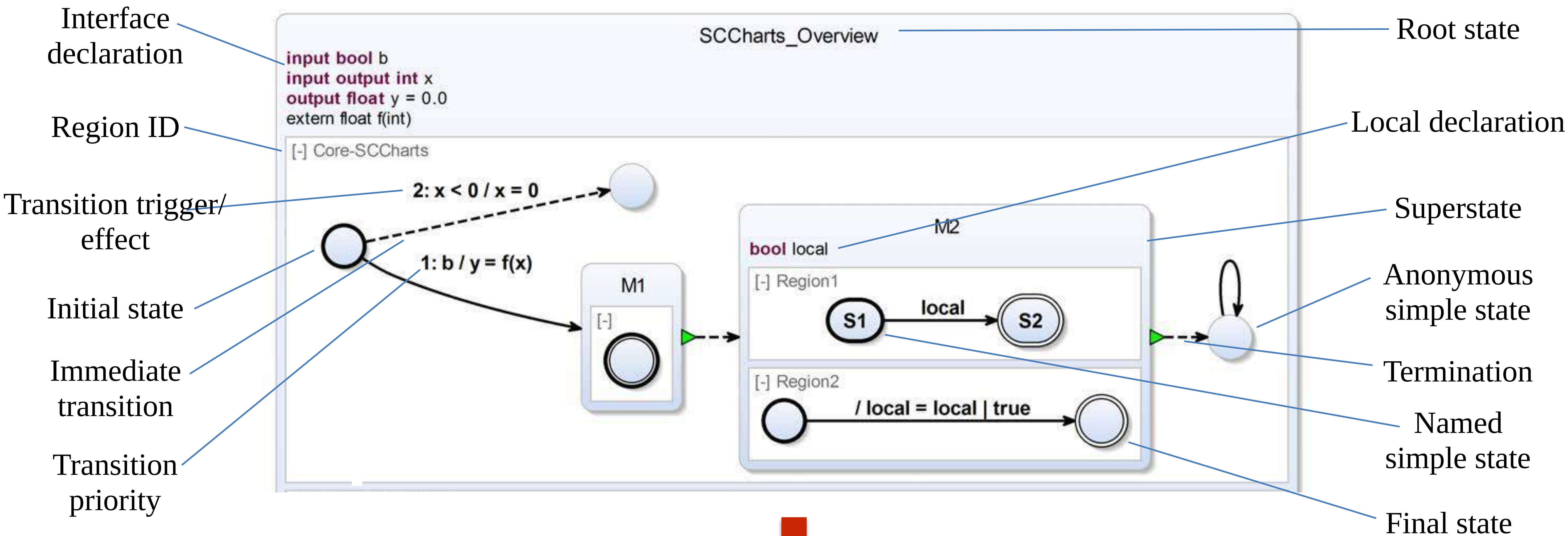


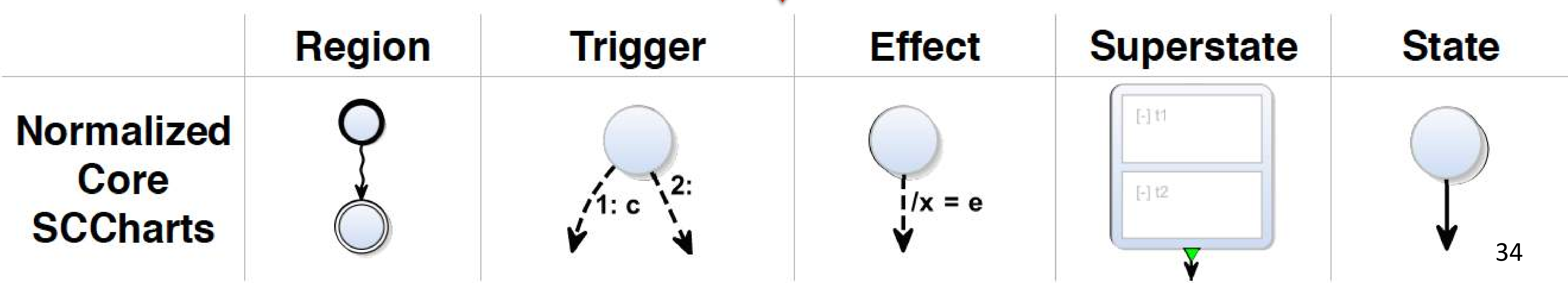
M2M Mappings

	Thread	Conditional	Assignment	Concurrency	Delay
SCL	*t*	if (*c*) s_1 else s_2	$x = e$	fork t_1 par t_2 join	pause
SCG					

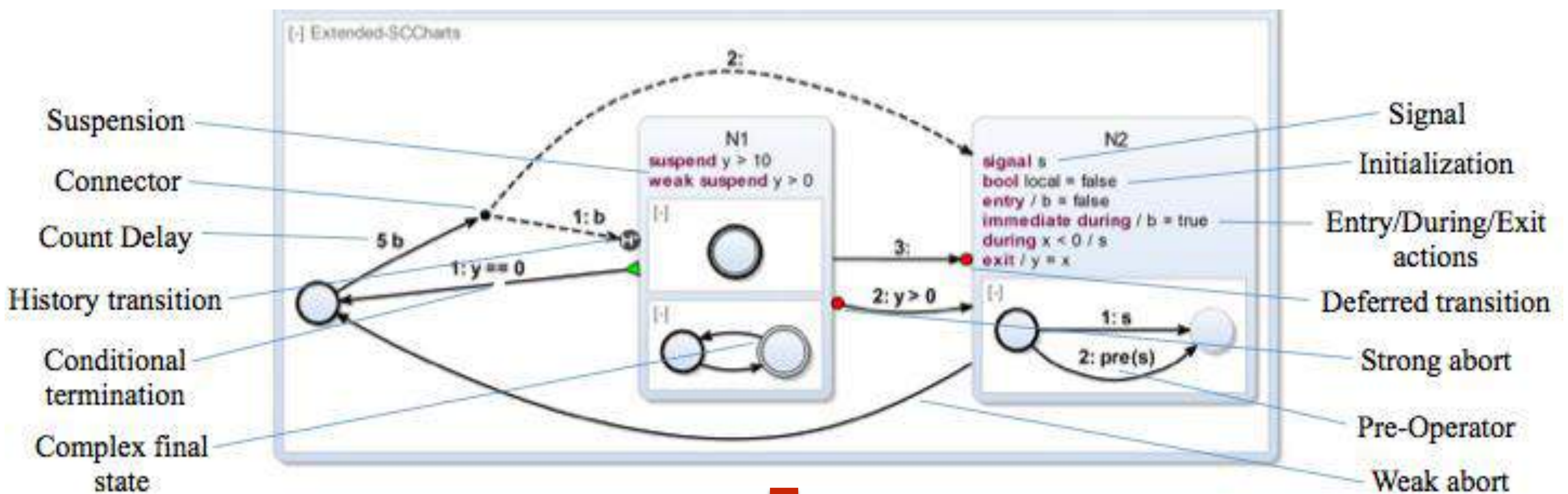


Some Syntactic Sugar: Core SCCharts

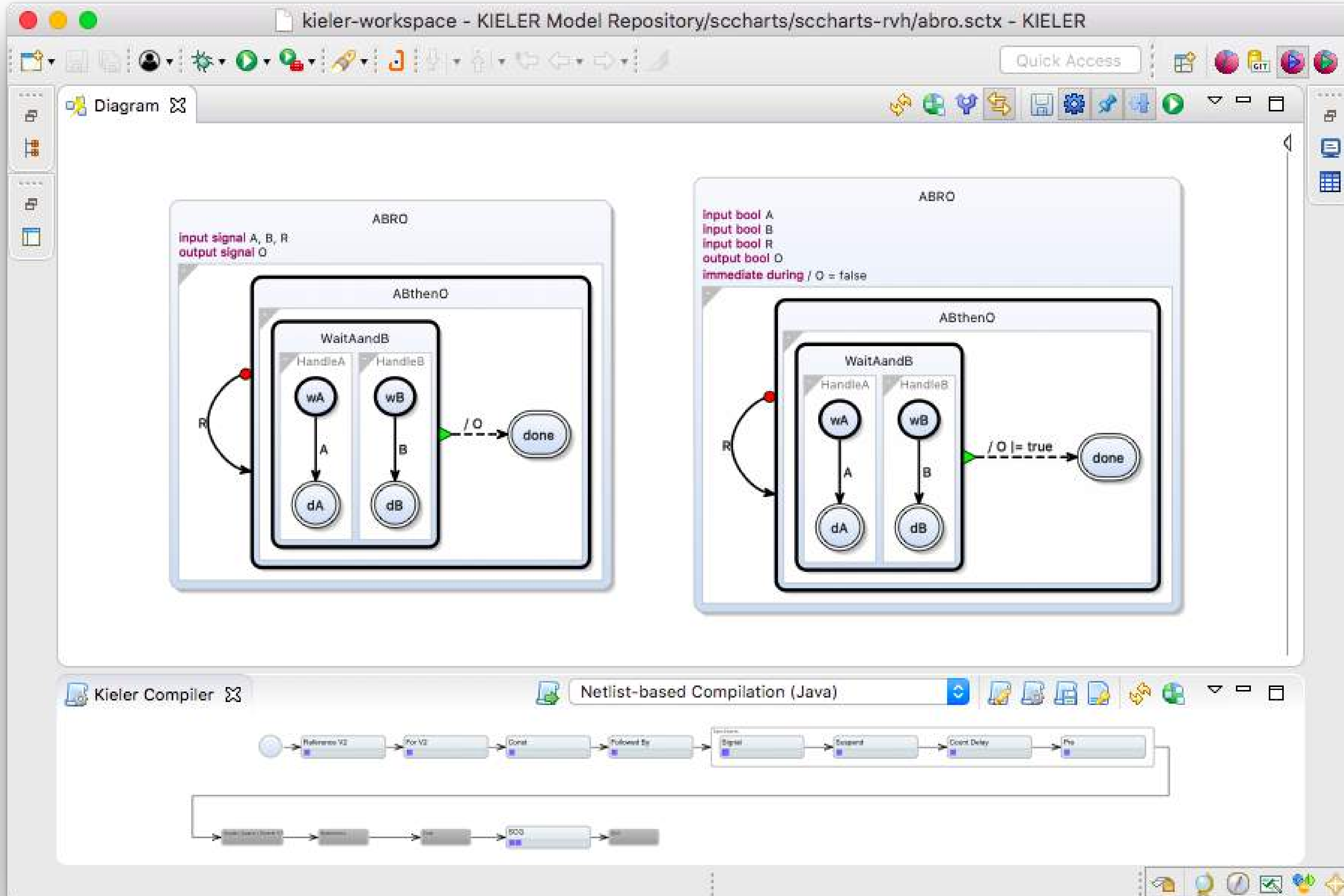




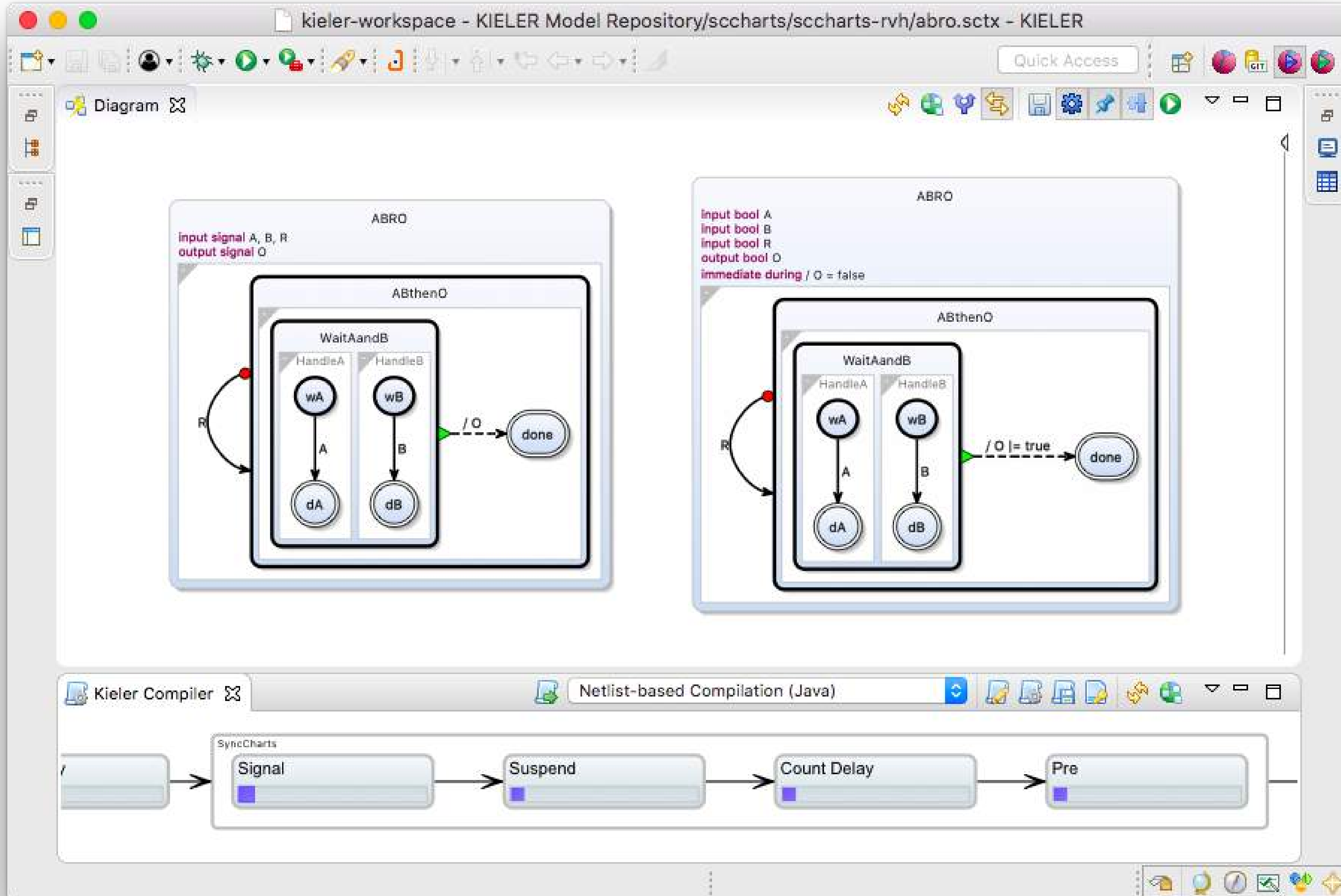
More Syntactic Sugar: Extended SCCharts



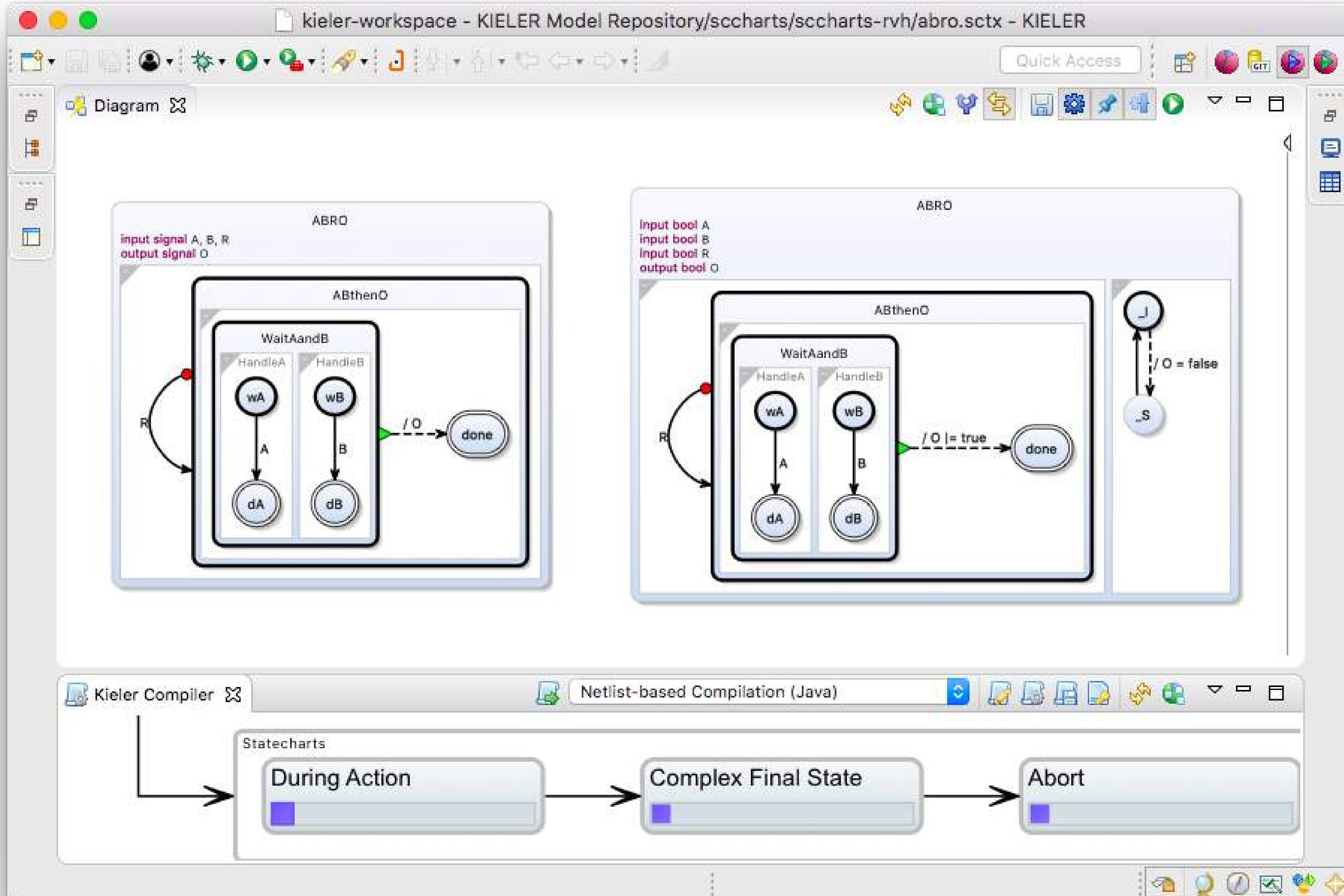
Compilation: Expand Signals



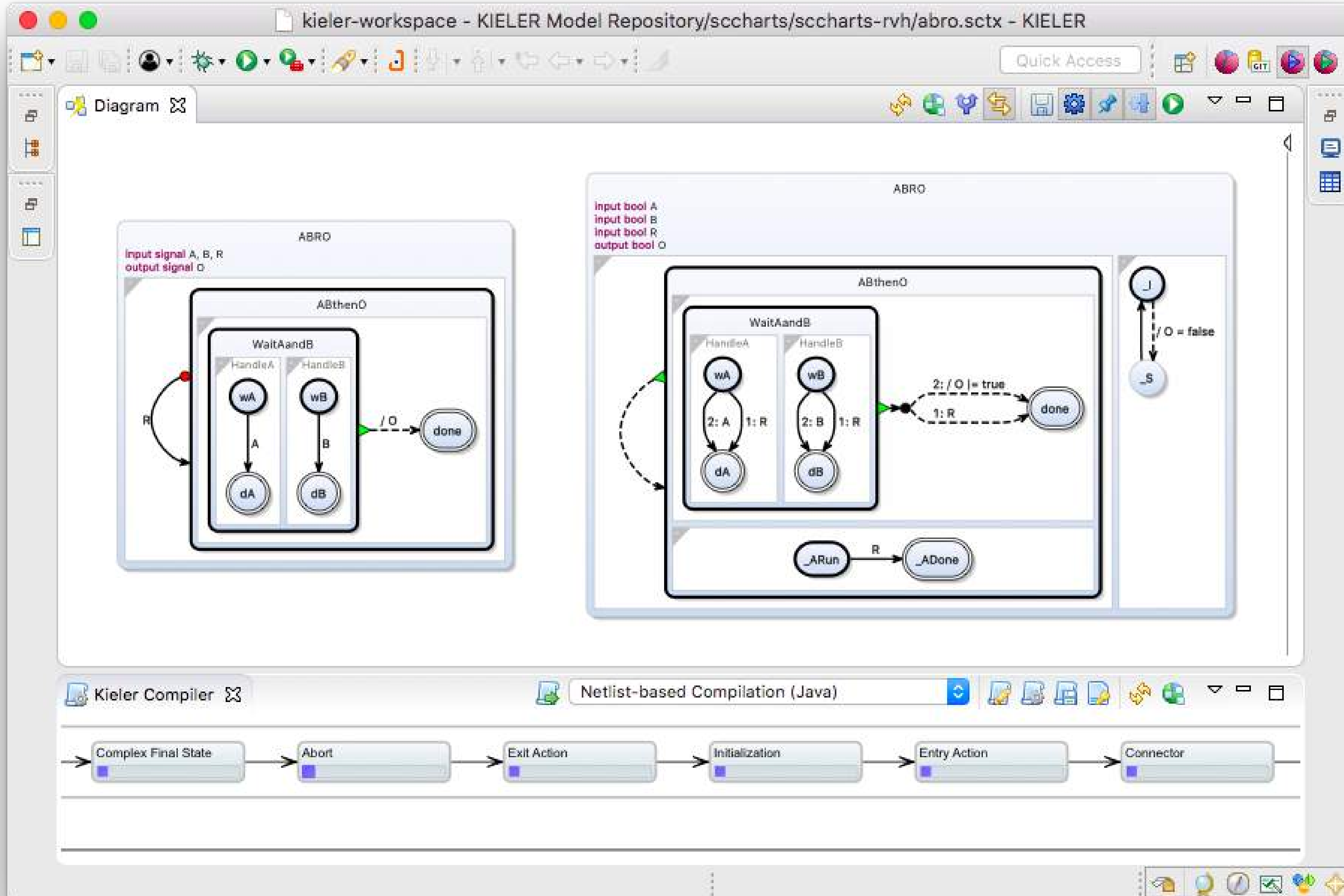
Compilation: Expand Signals



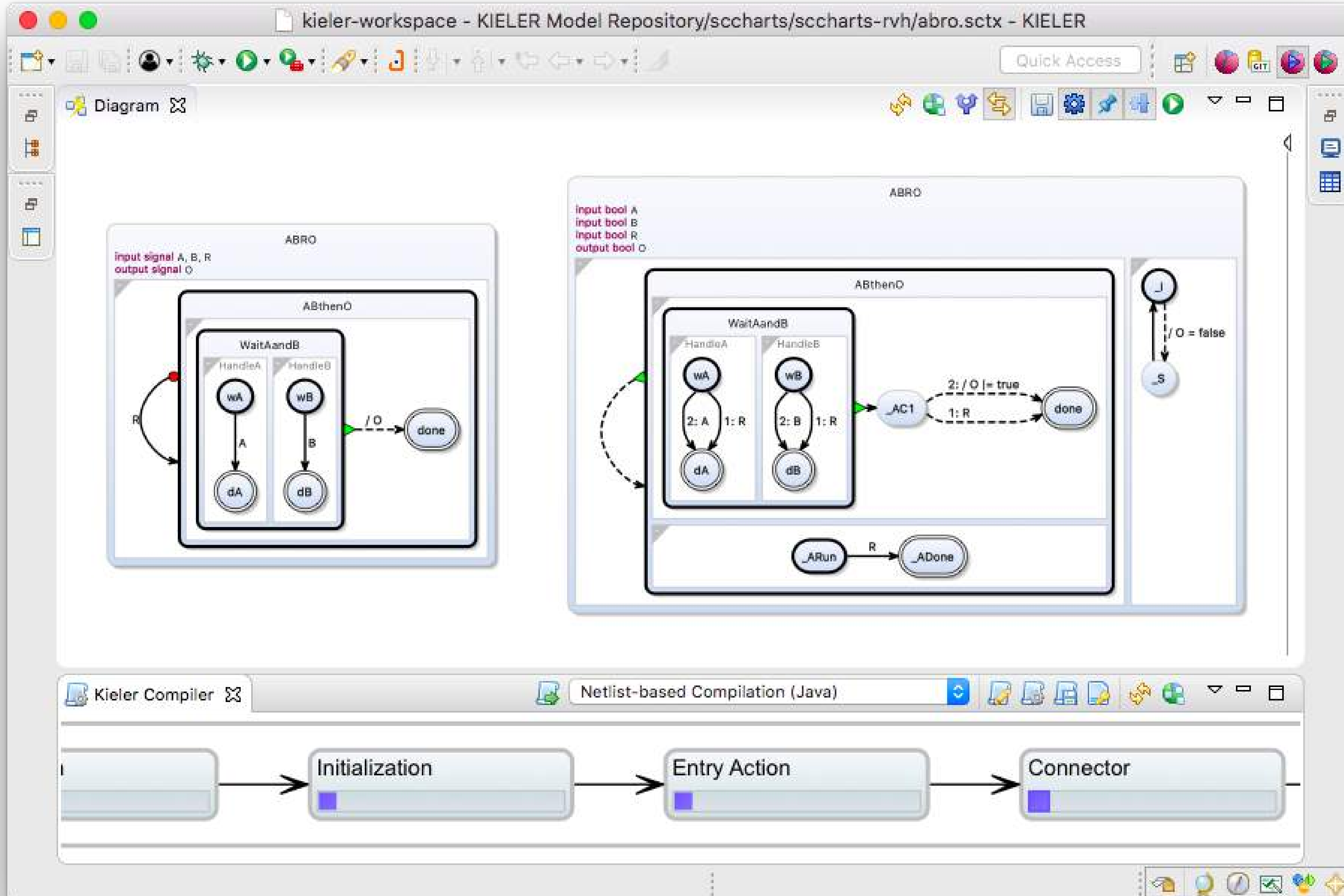
Expand During Action



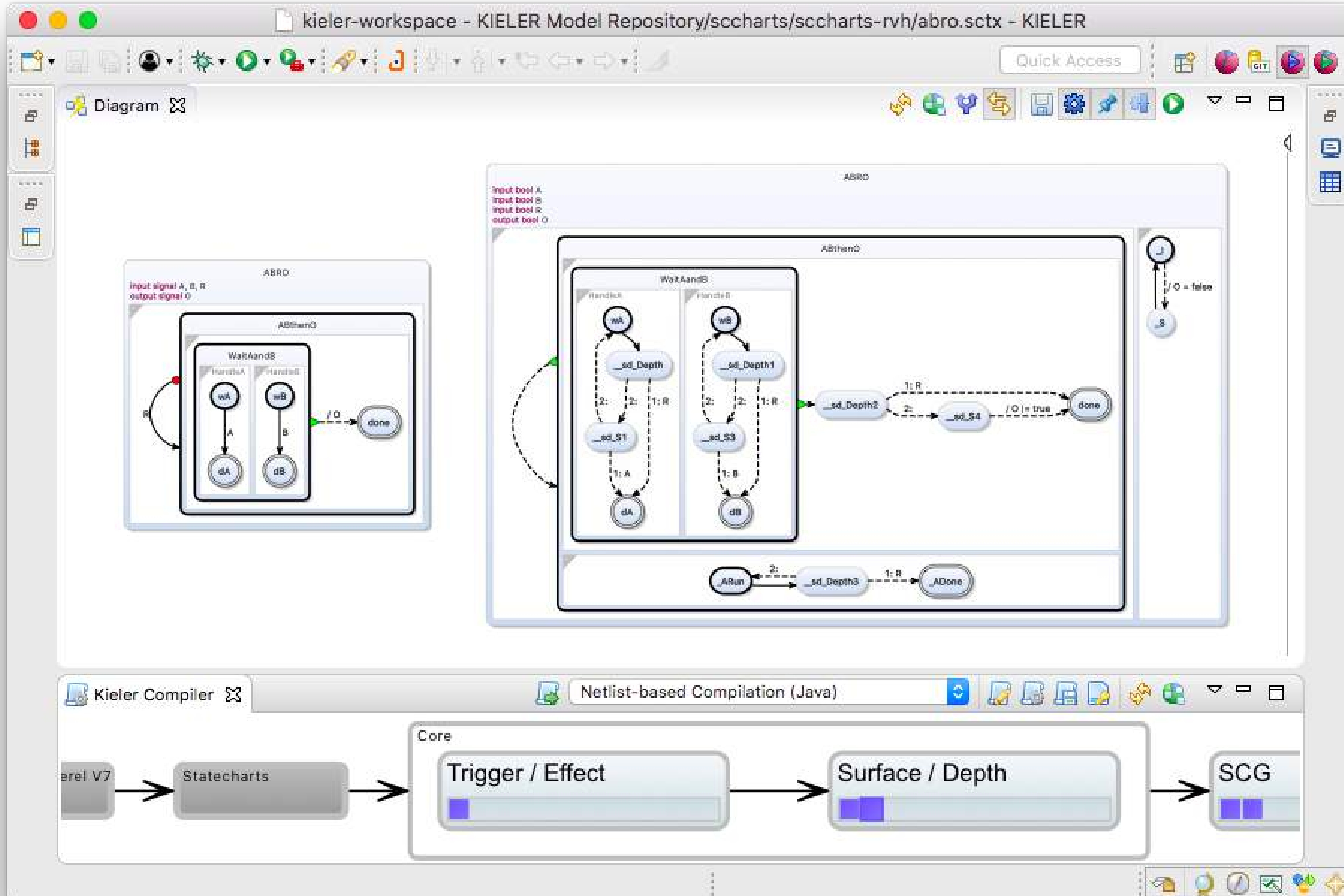
Expand Abort



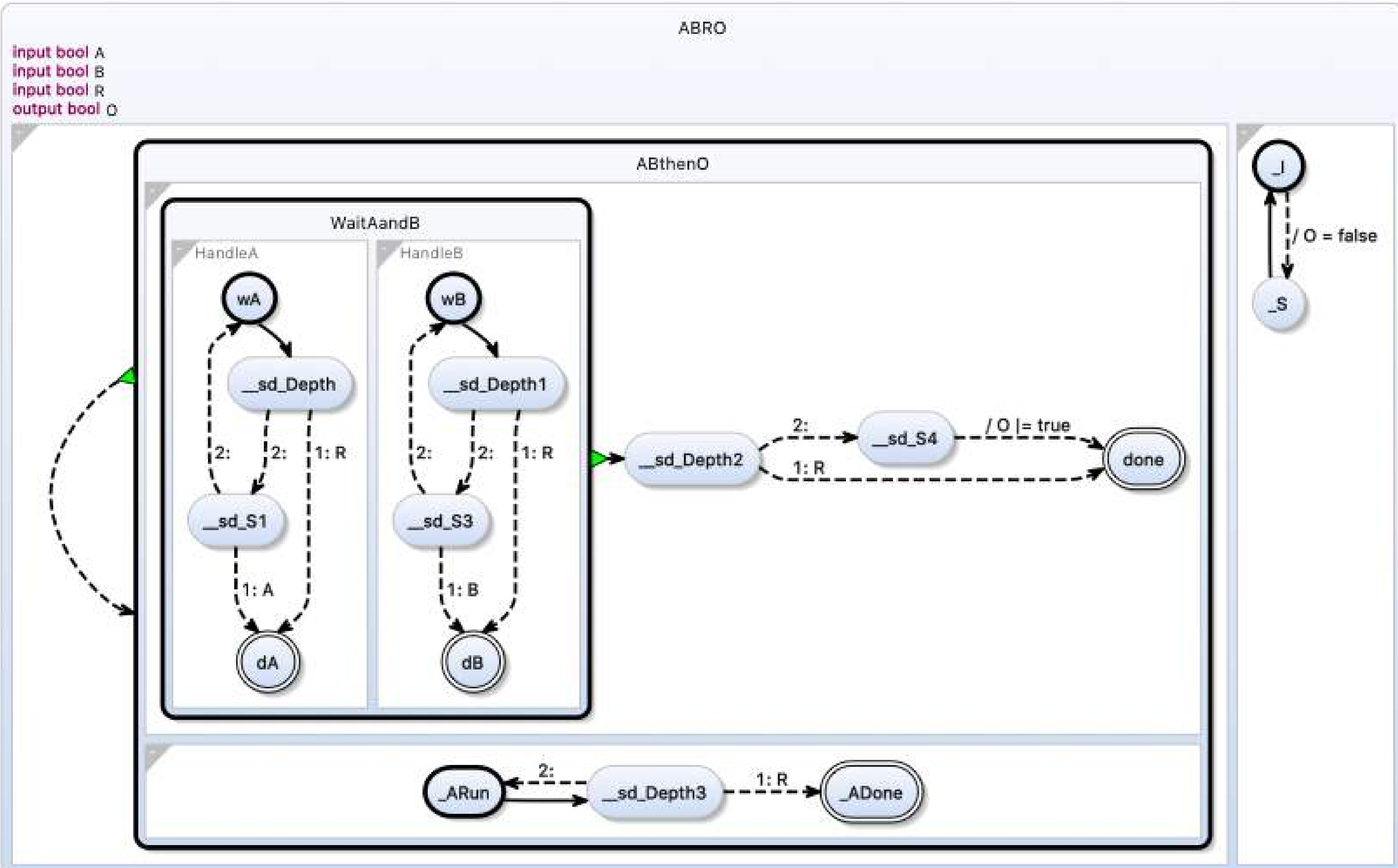
Core SCChart



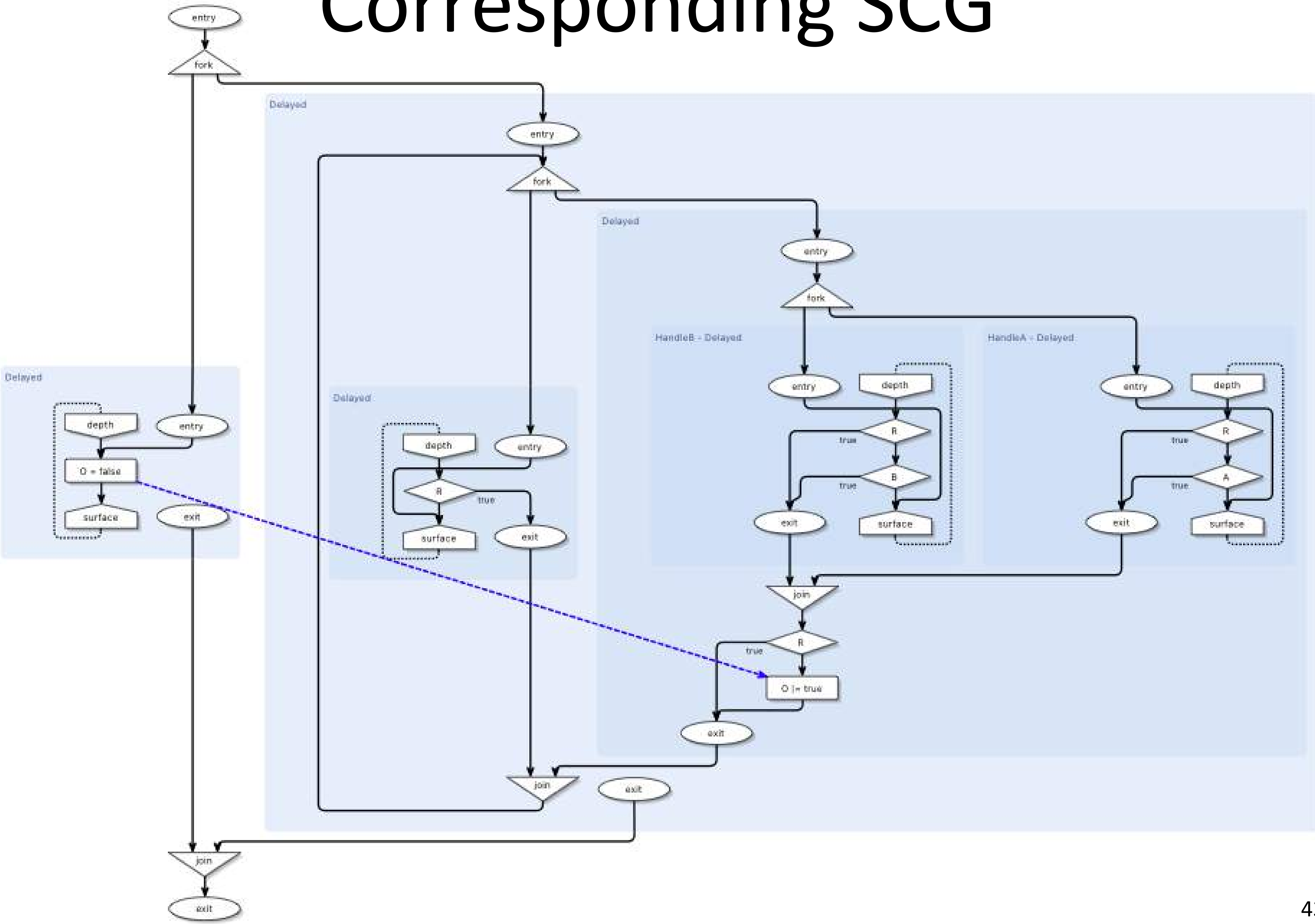
Normalized Core SCChart



Normalized Core SCChart



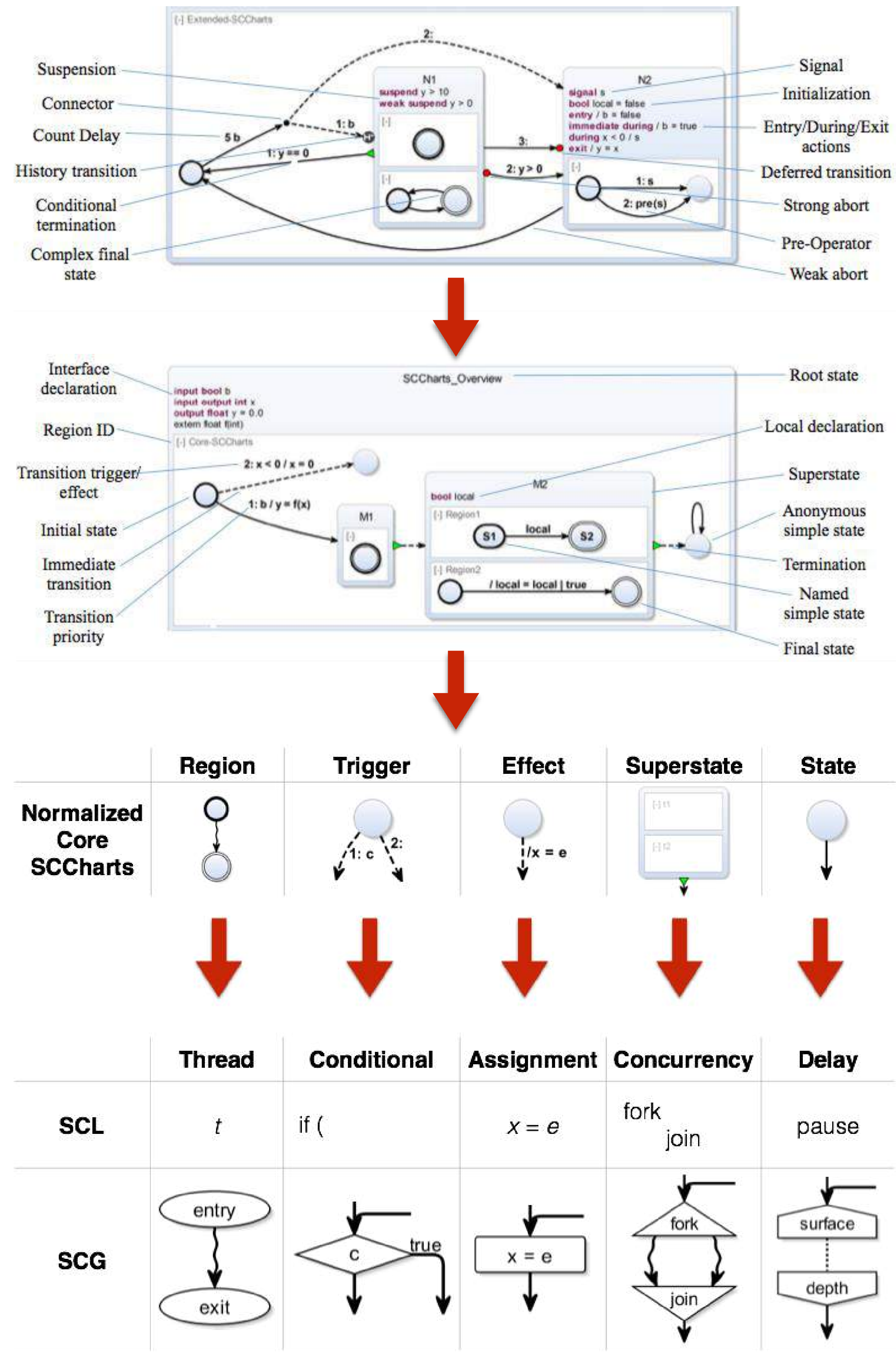
Corresponding SCG



Taking Stock

SCCharts defined by M2M Transformations

- Extended SCCharts
- Core SCCharts
- Normalized Core SCCharts
- SCL/SCG



SCCharts – Pragmatics

Key to Pragmatics: MVC

- A **model** represents knowledge

- A **view** is a (visual) representation of its model.
  It would ordinarily highlight certain attributes of the model and suppress others.
  It is thus acting as a **presentation filter**.

- A **controller** is the link between a user and the system.
  It provides the user with input by **arranging for relevant views to present themselves in appropriate places on the screen**.

[Trygve Reenskaug,
Models – Views – Controllers,
Xerox PARC technical note, 1979]

Textual Modeling

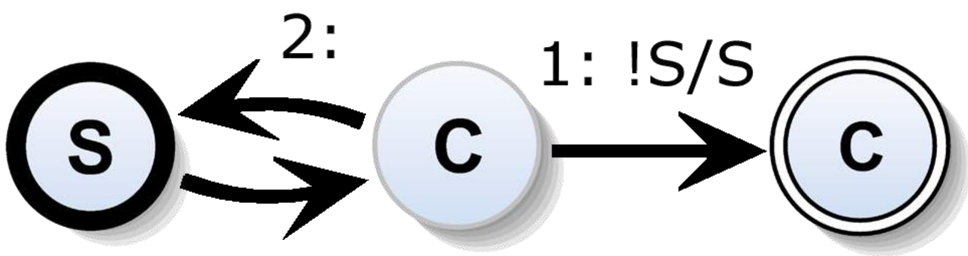
SCChart **model** specified in .sctx

- Efficient editing
- Facilitates model comparison
- Easy revision control

Graphical **view** automatically synthesized with KIELER (**controller**)

- Customizable view
- Saves developer time

```
scchart ABRO {
 input signal A, B, R
 output signal O


 initial state ABthenO {


  initial state WaitAandB {
   region "HandleA" {
    initial state wA
    if A go to dA


    final state dA
   }

   region "HandleB" {
    initial state wB
    if B go to dB


    final state dB
   }
  }
  join to done do O


  final state done
 }
 if R abort to ABthenO
}
```



SCCharts

http://www.sccharts.com/

KIELER

The Key to Efficient Modeling

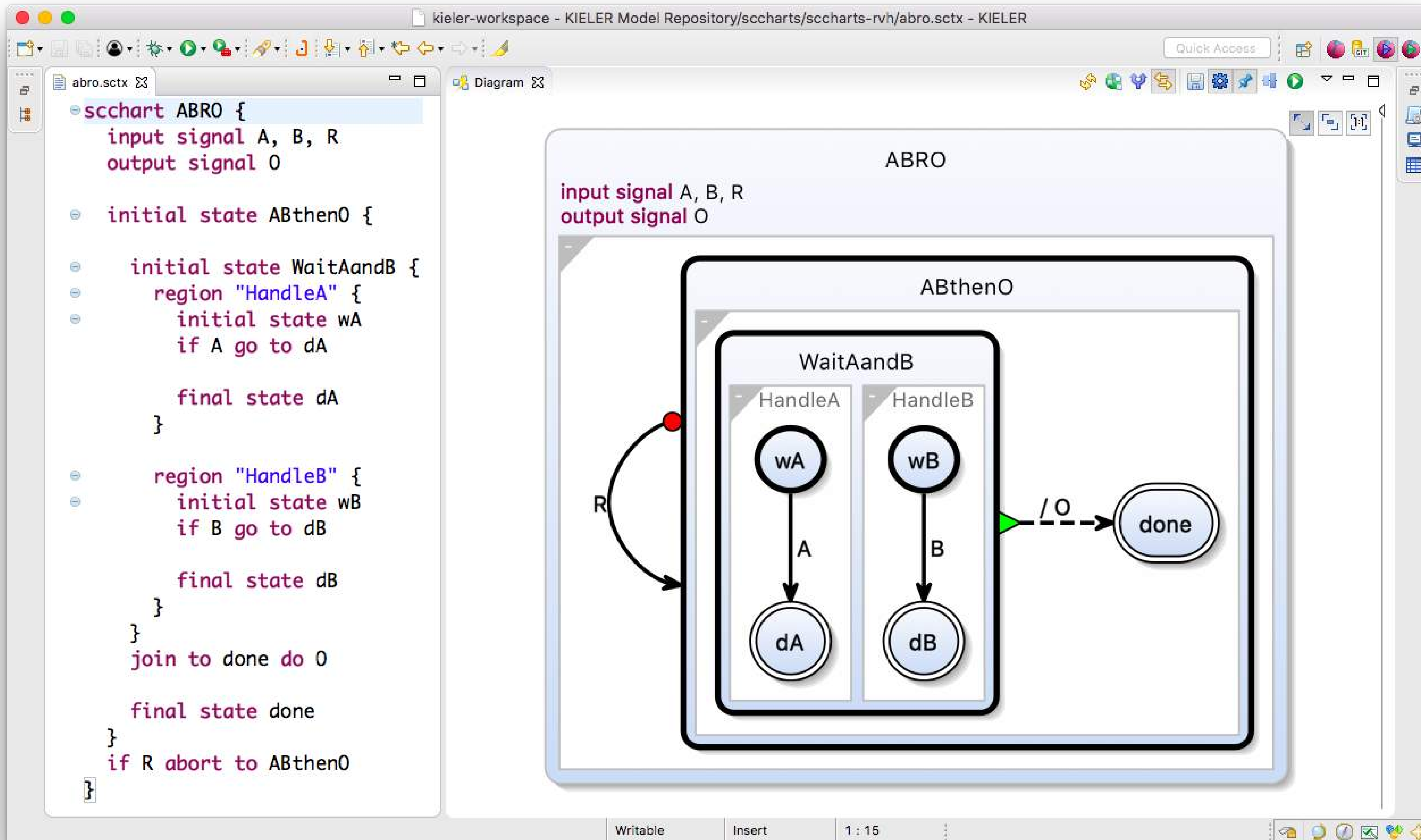
http://www.rtsys.informatik.uni-kiel.de/en/research/kieler

Eclipse Layout Kernel

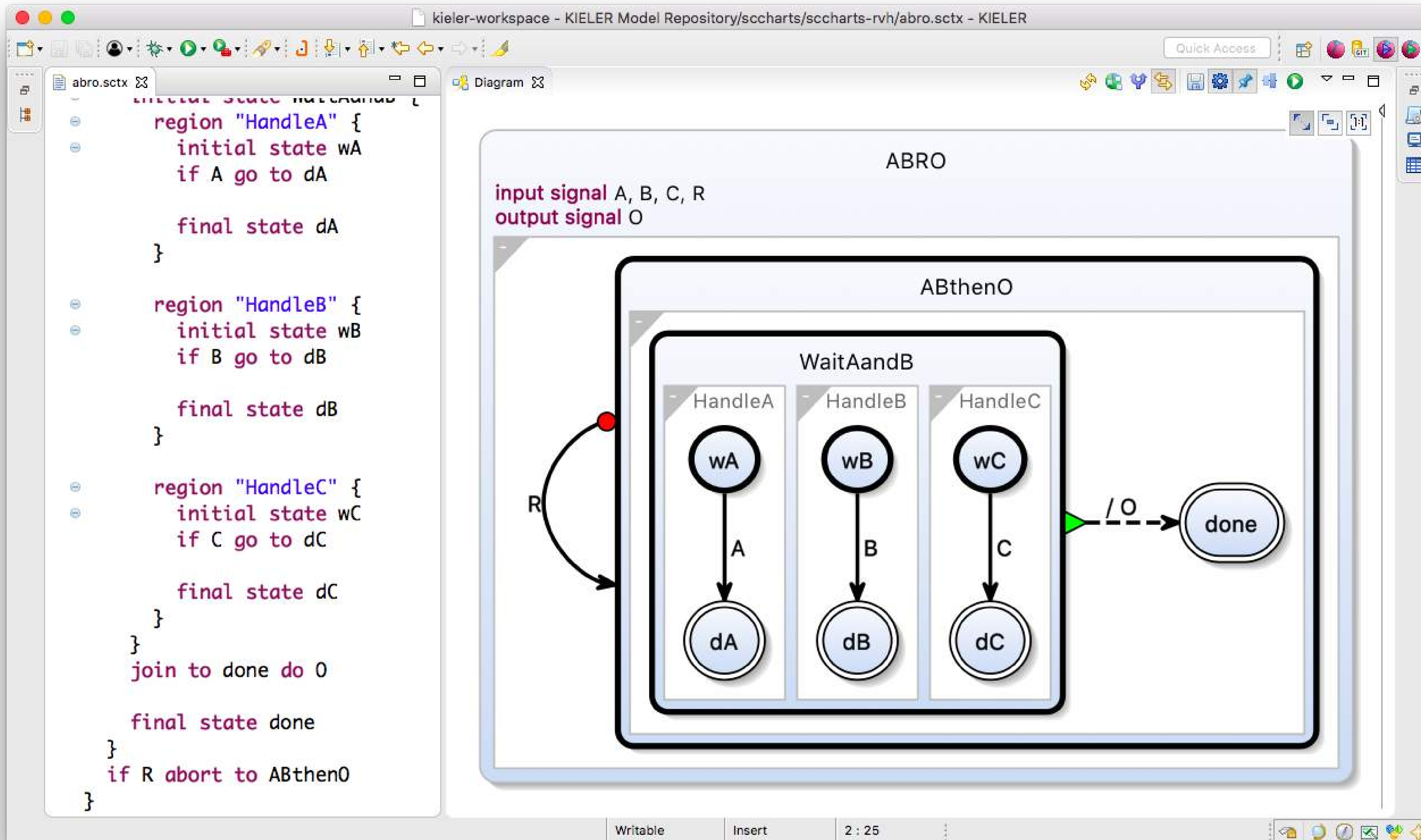
https://www.eclipse.org/elk/

All open source, under EPL license

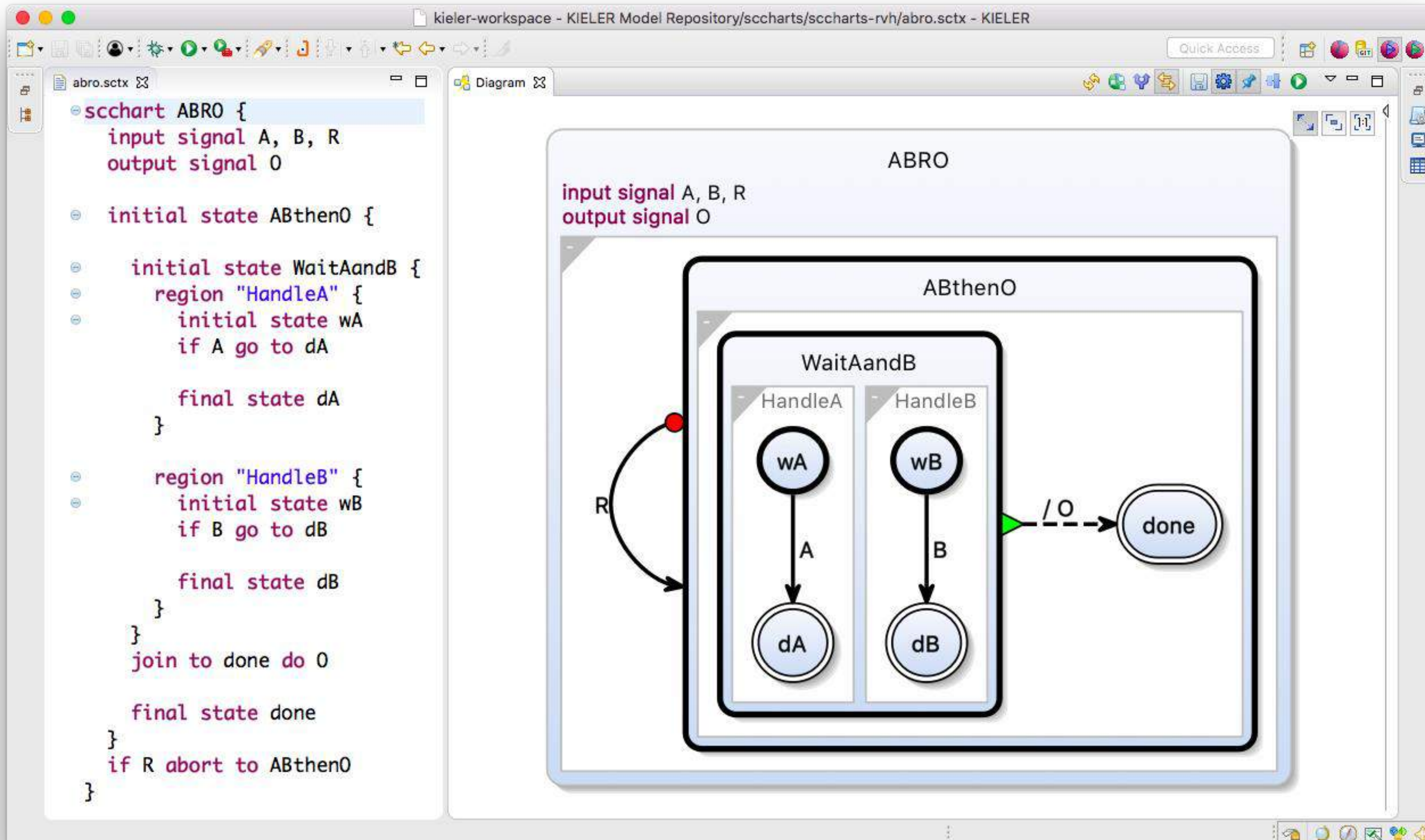
View Synthesis



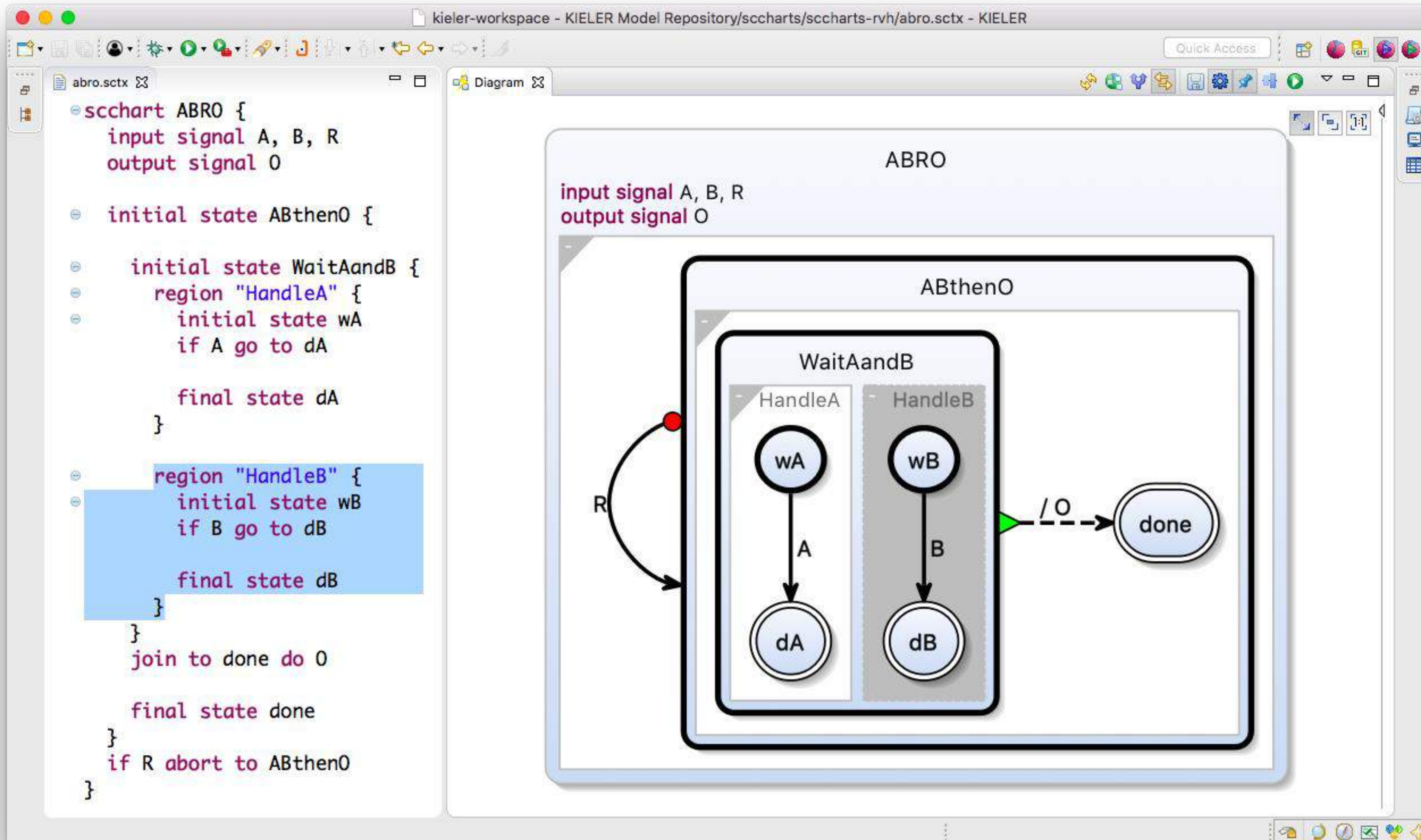
Adding Region "HandleC"



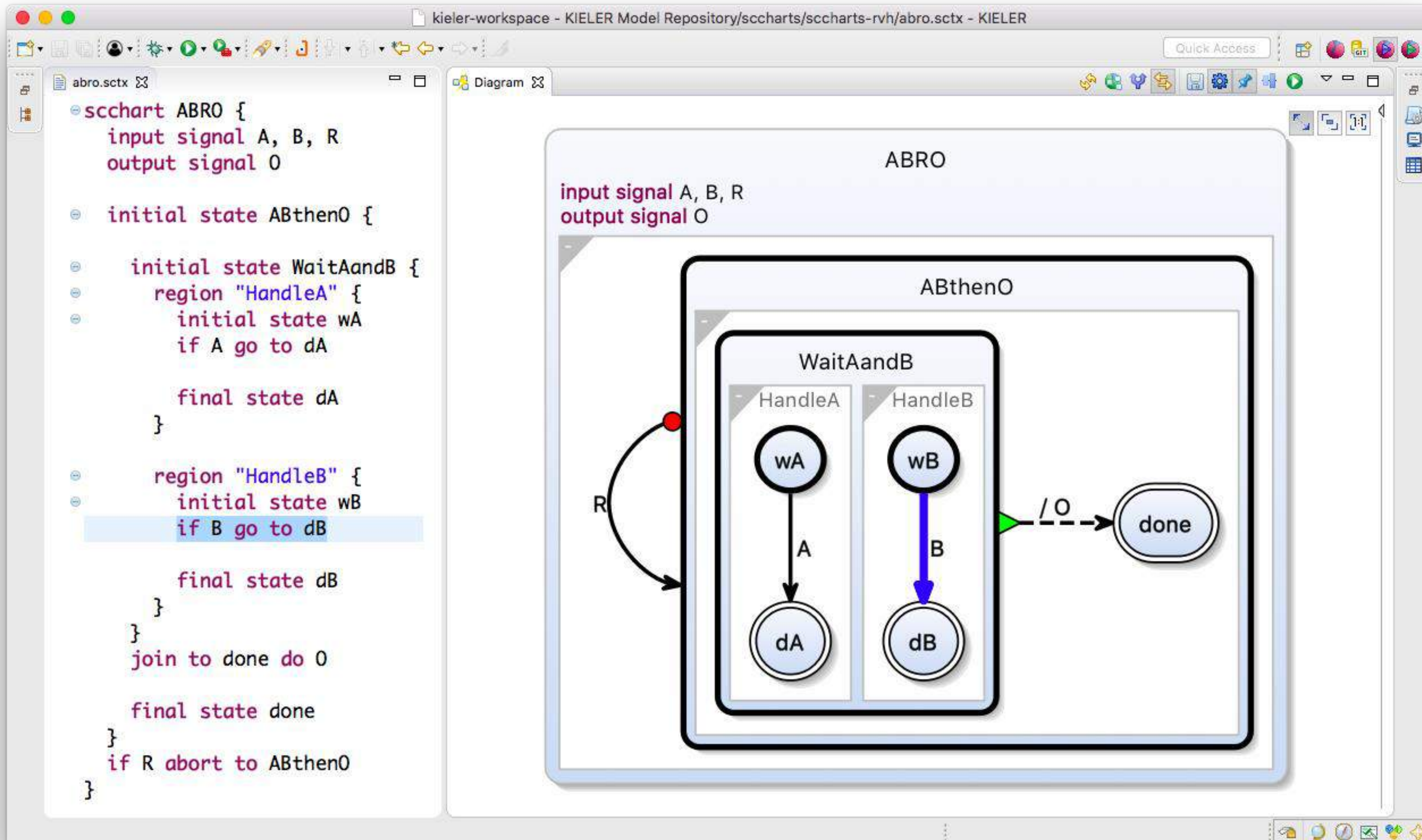
Back to Original ABRO



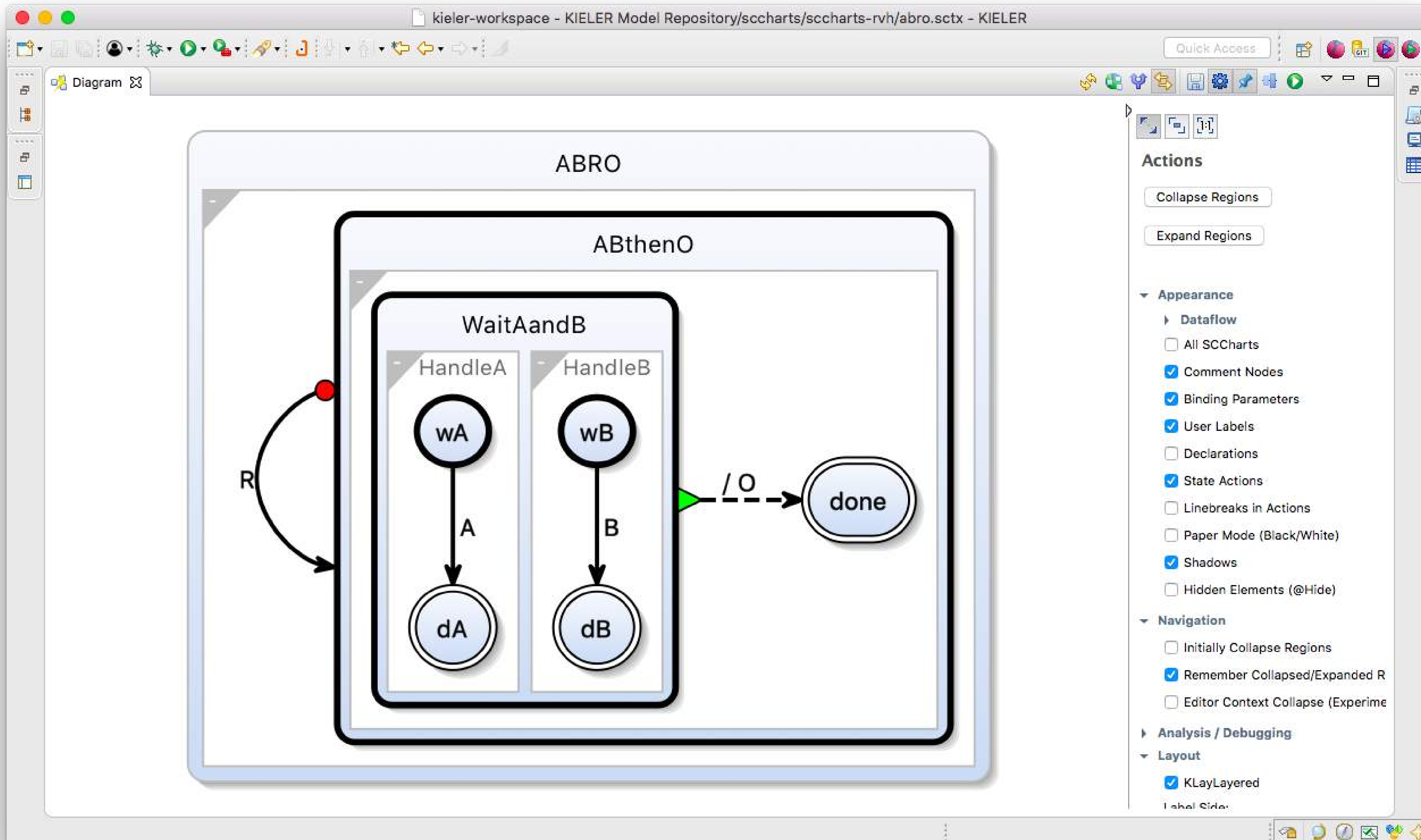
Graphics-to-Text Navigation: Select Region



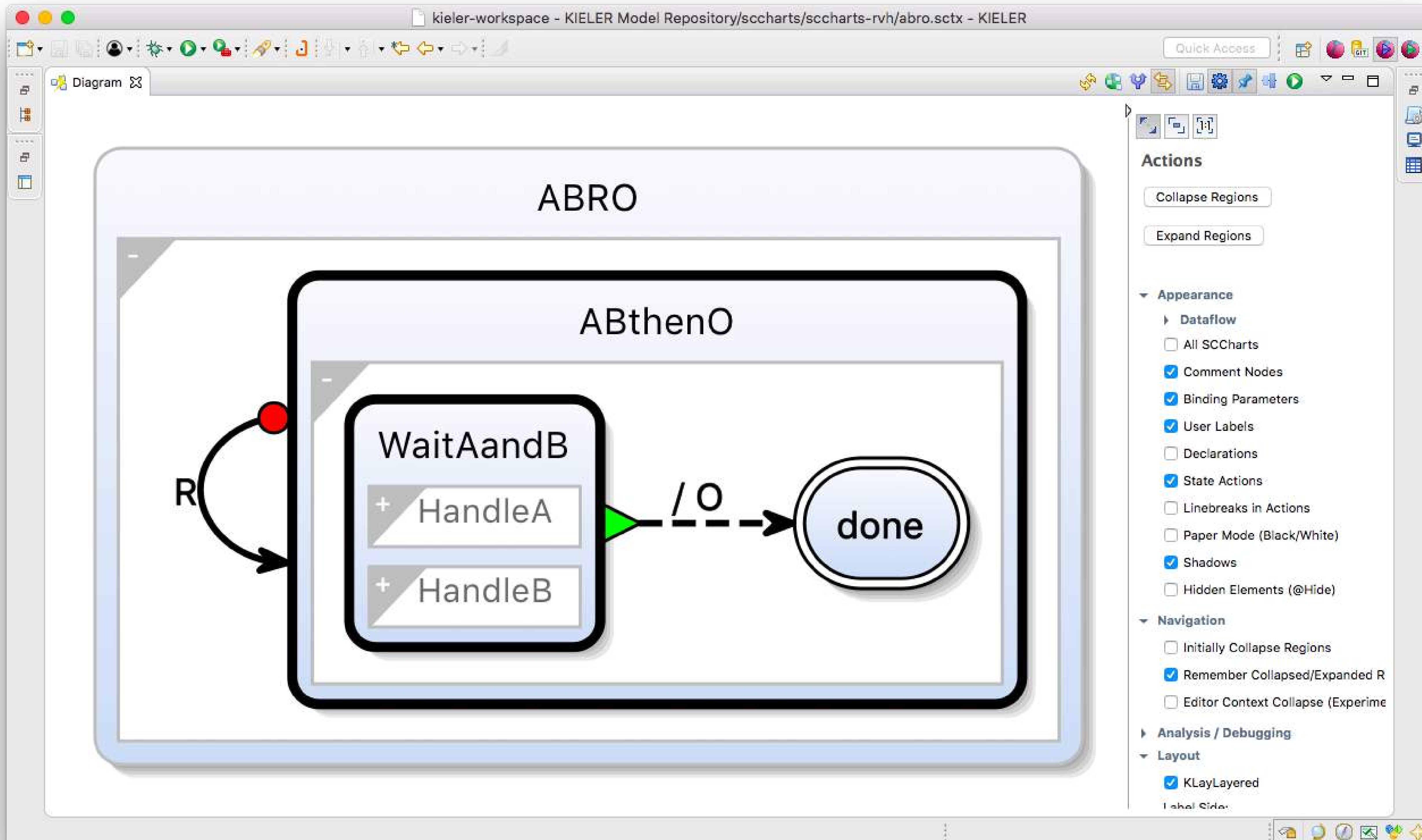
Graphics-to-Text Navigation: Select Transition



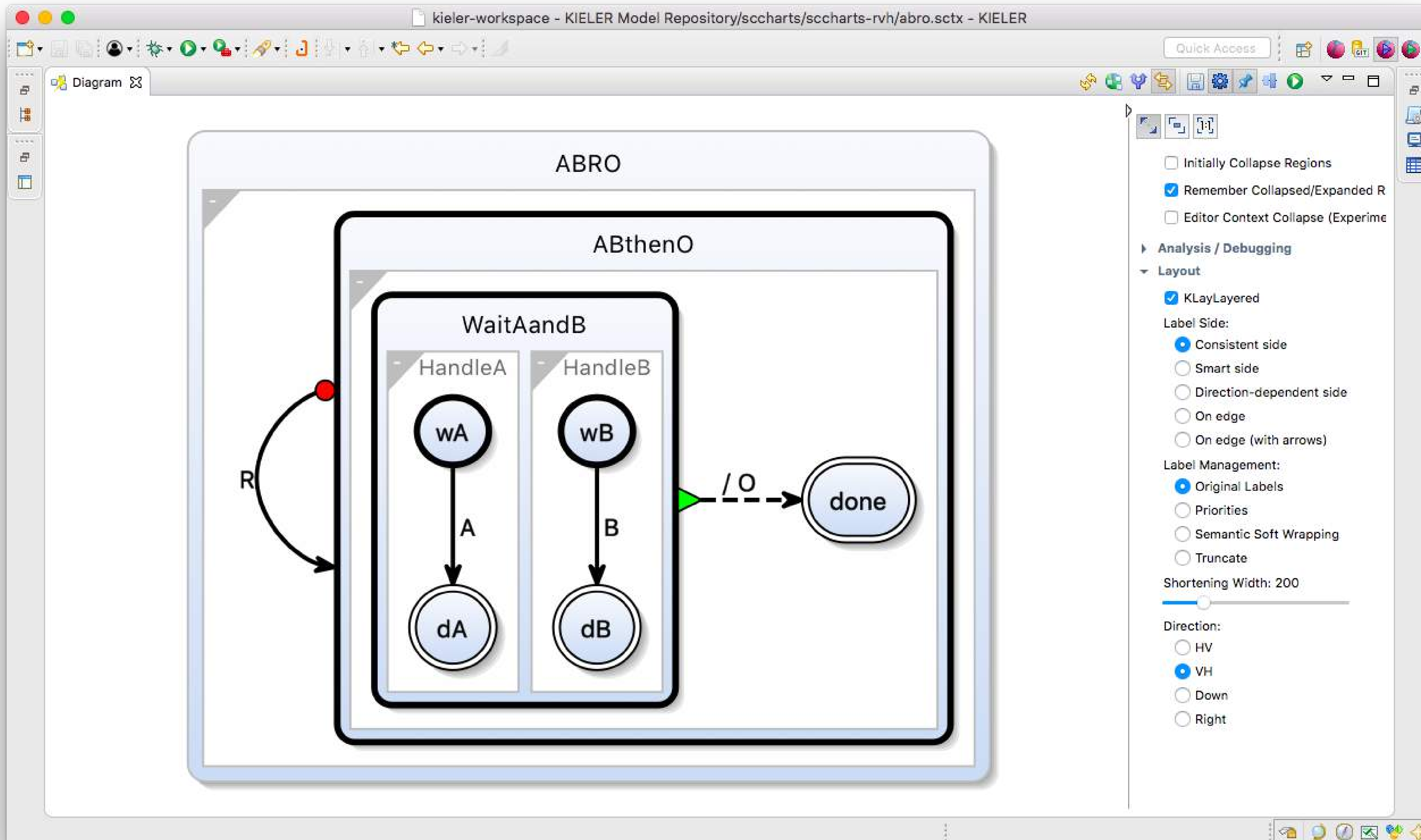
View Filtering: Hide Declarations



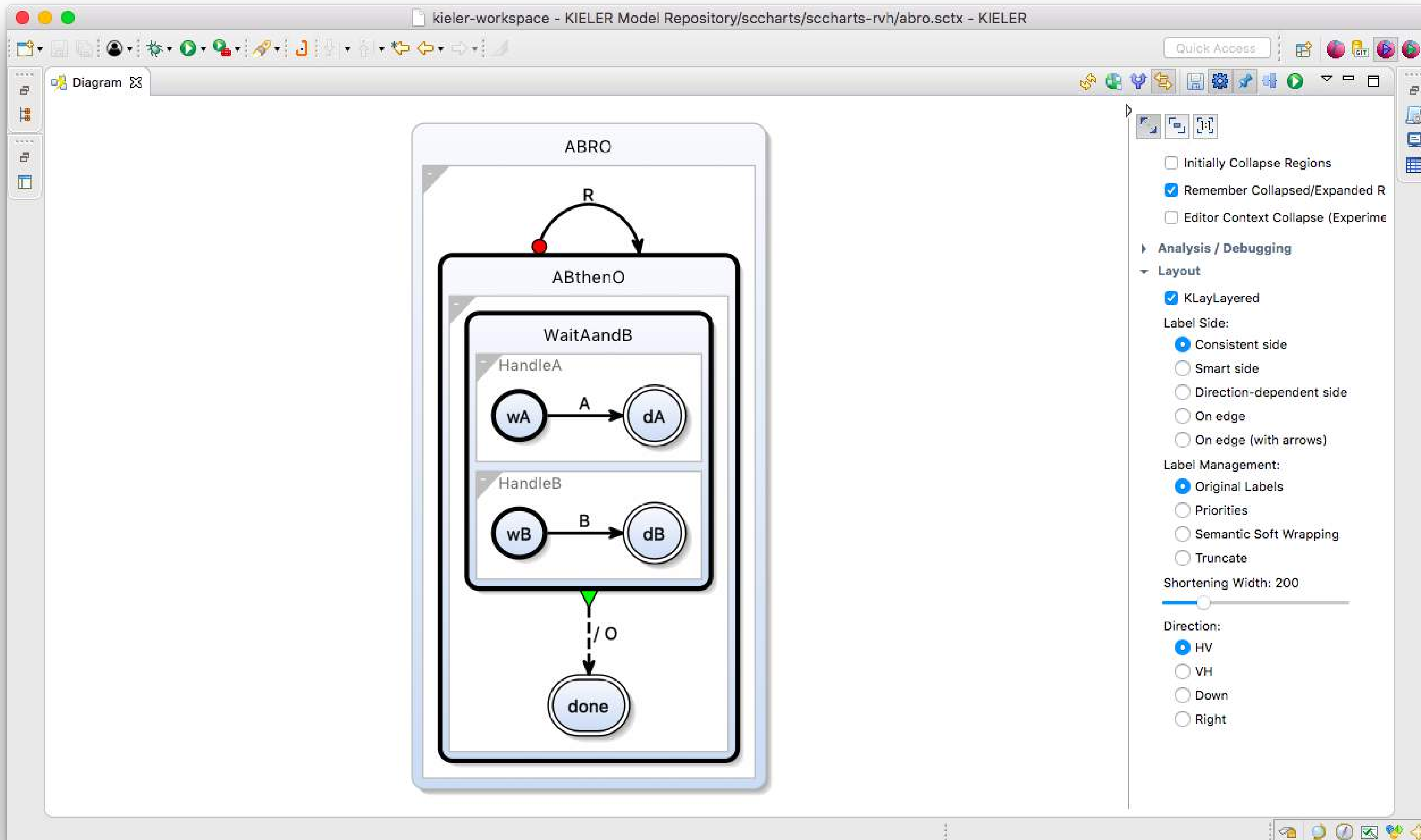
View Filtering: Collapse Regions



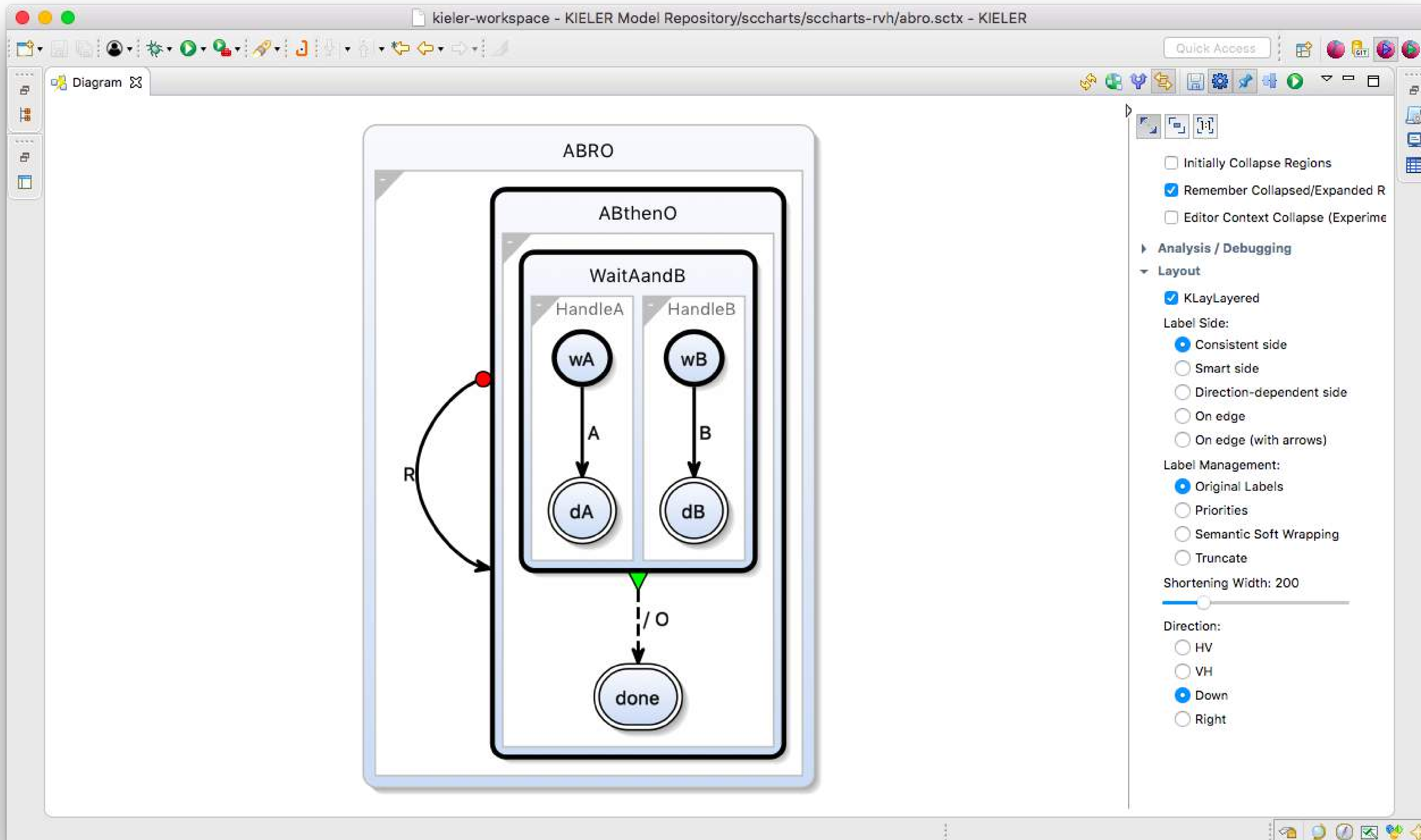
Direction VH (Vertical-Horizontal)



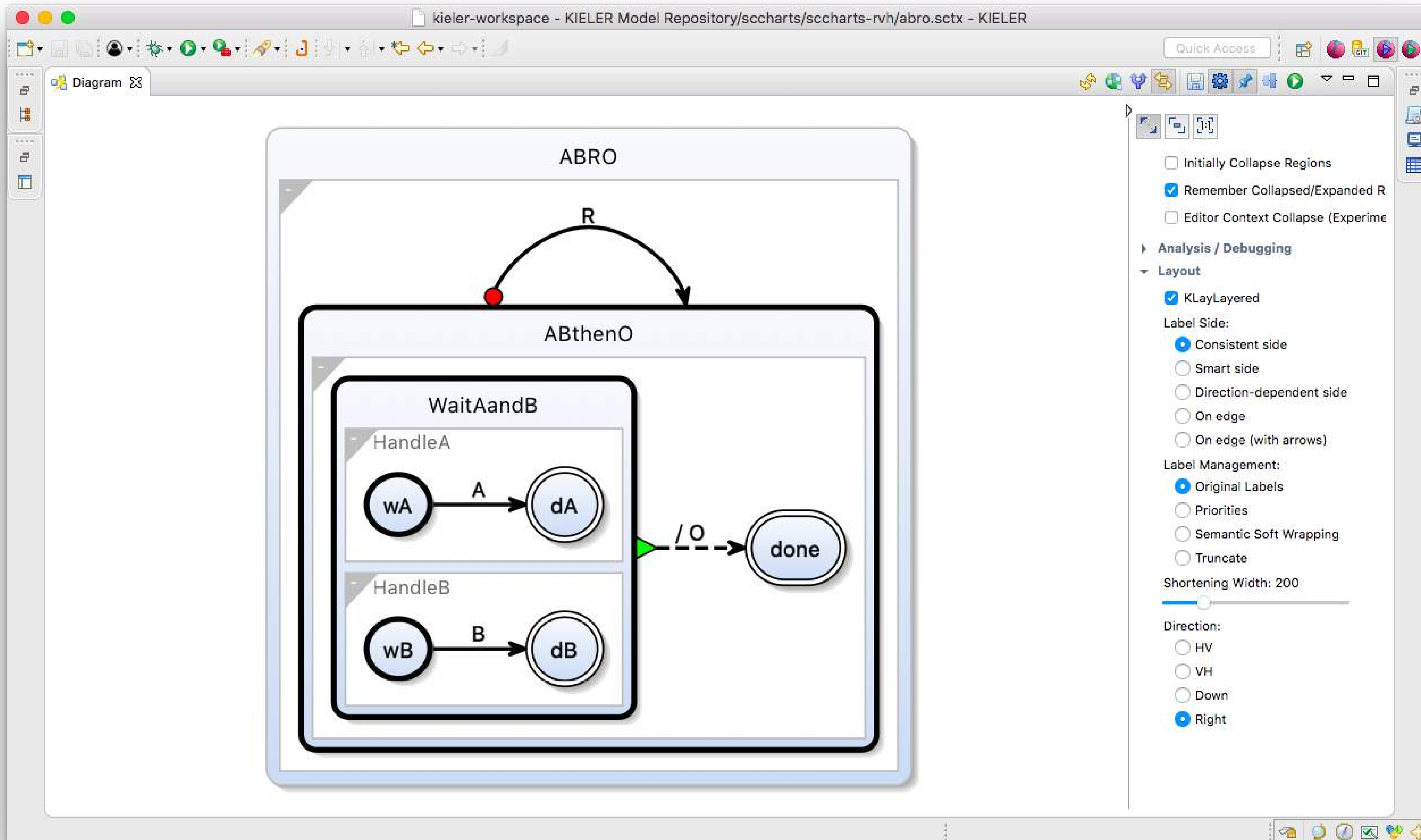
Change Layout: Direction HV



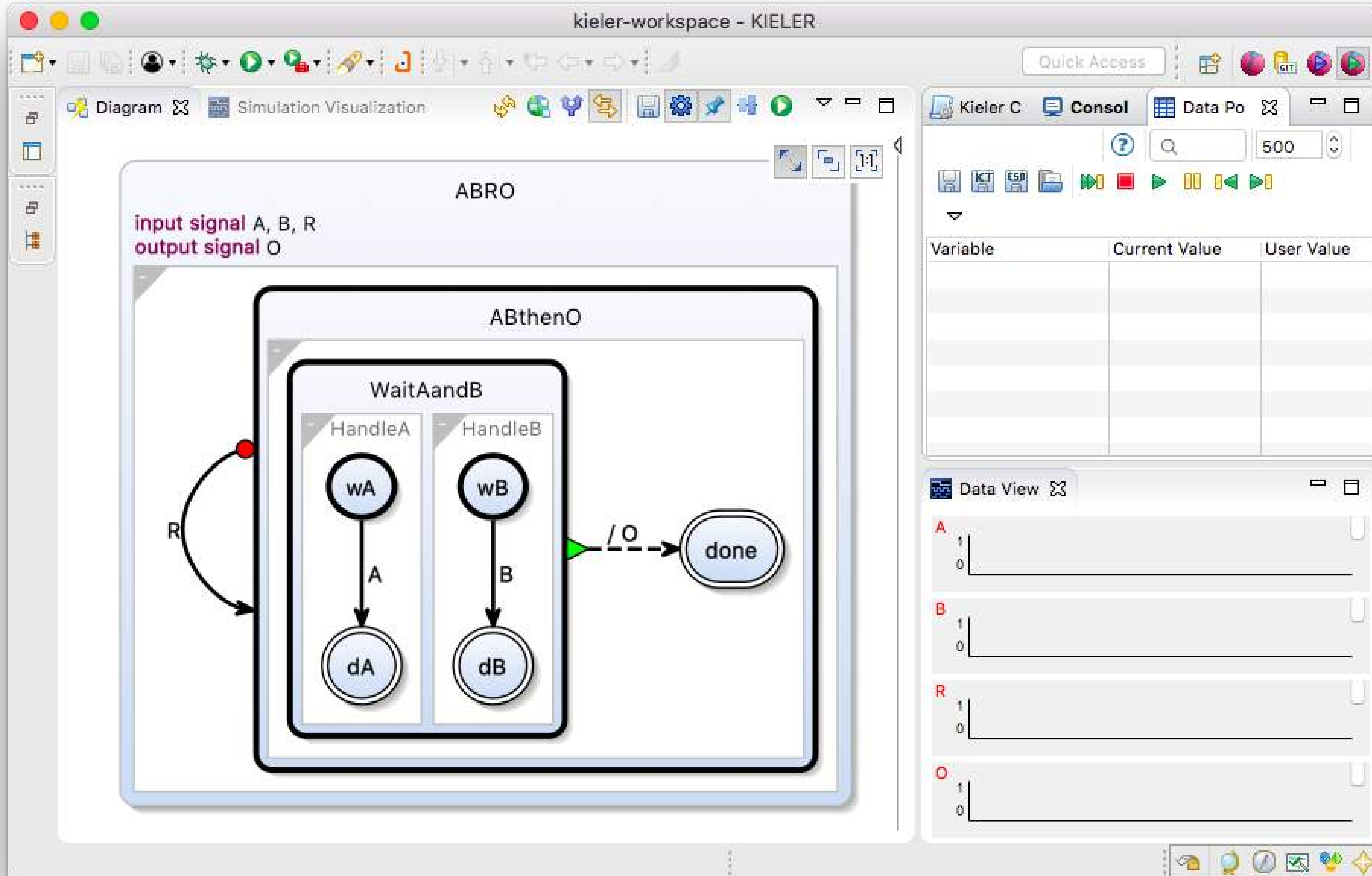
Change Layout: Direction Down



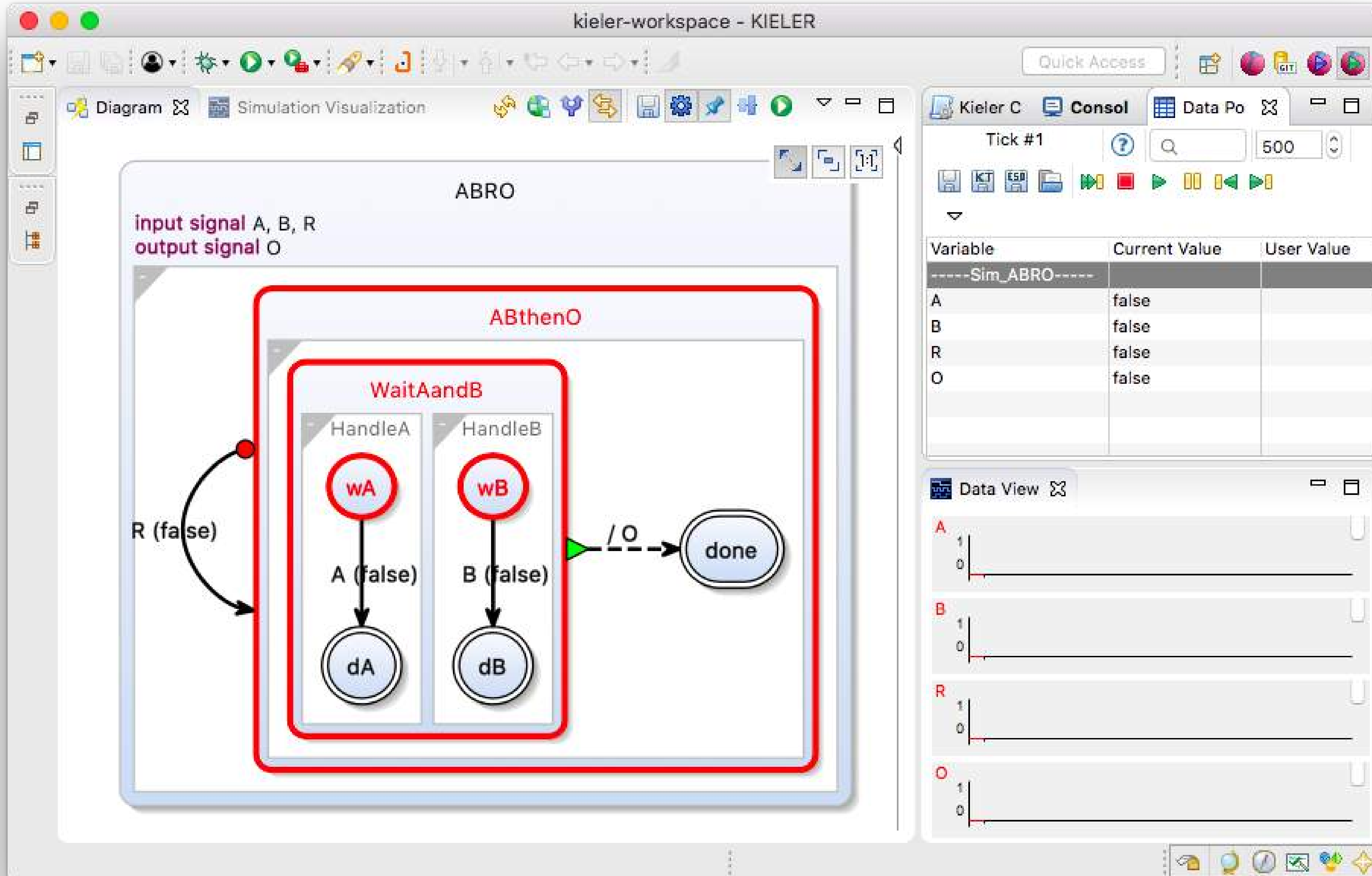
Change Layout: Direction Right



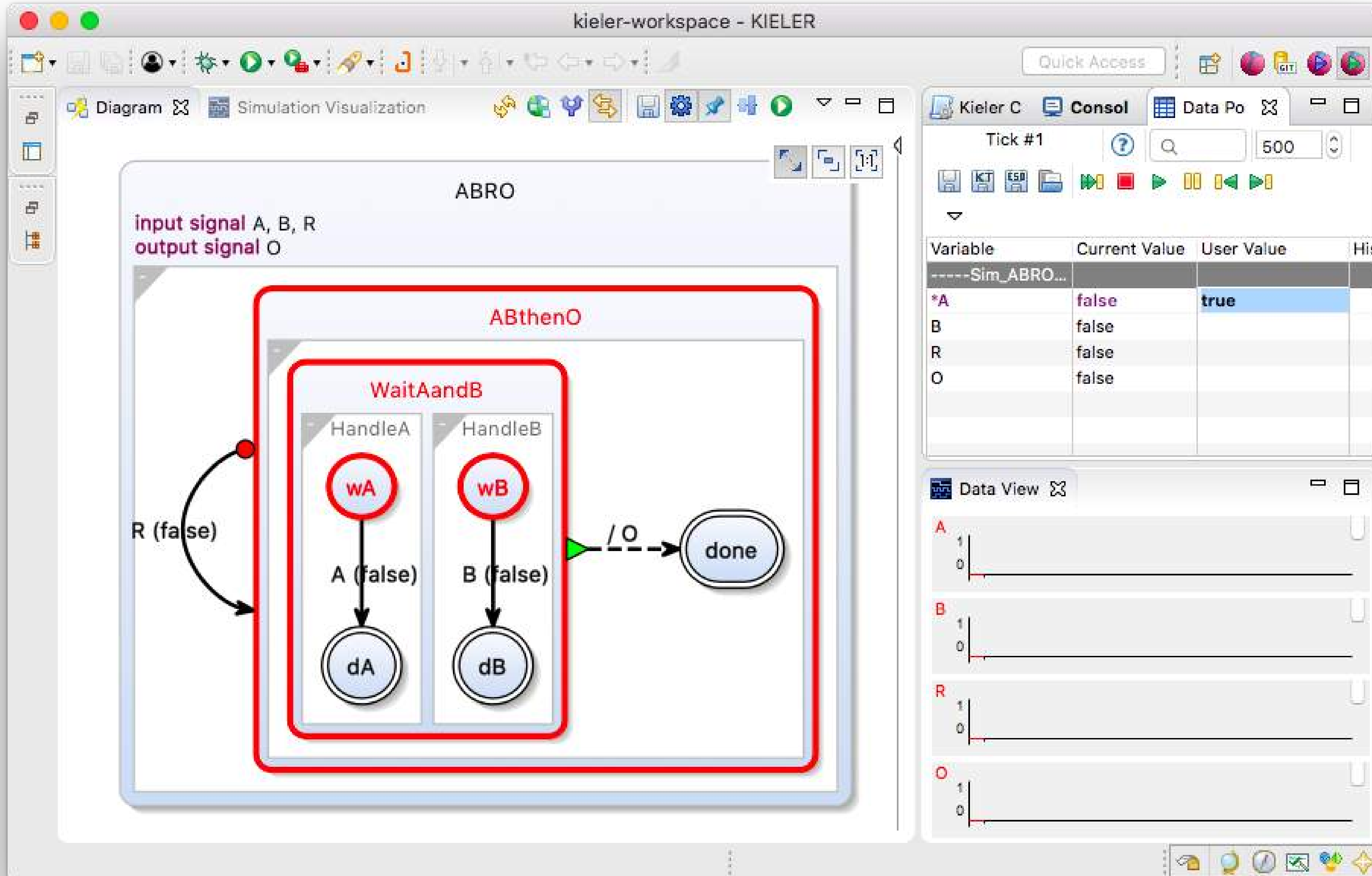
Simulation – Initialization



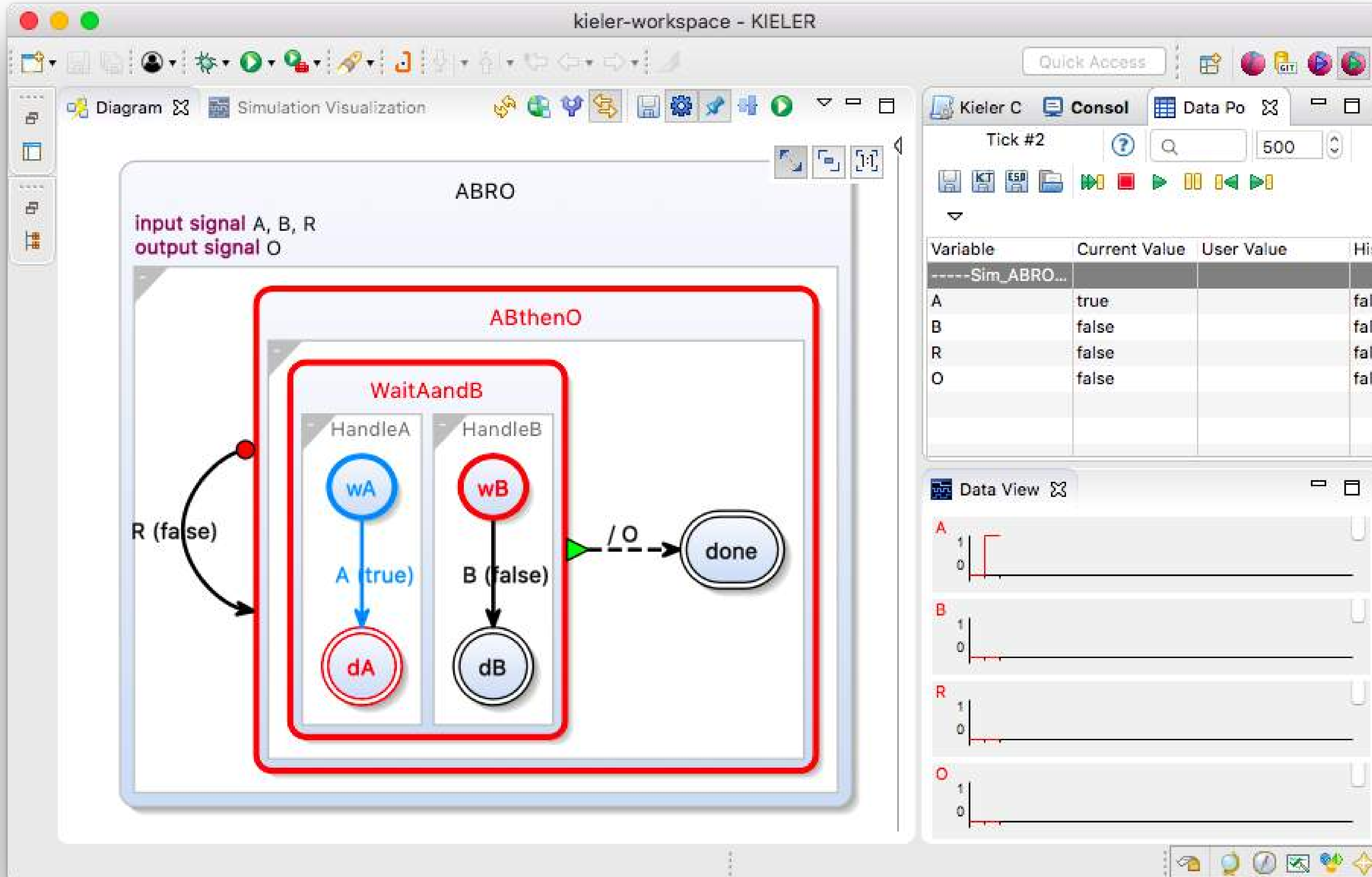
Simulation – Tick 1



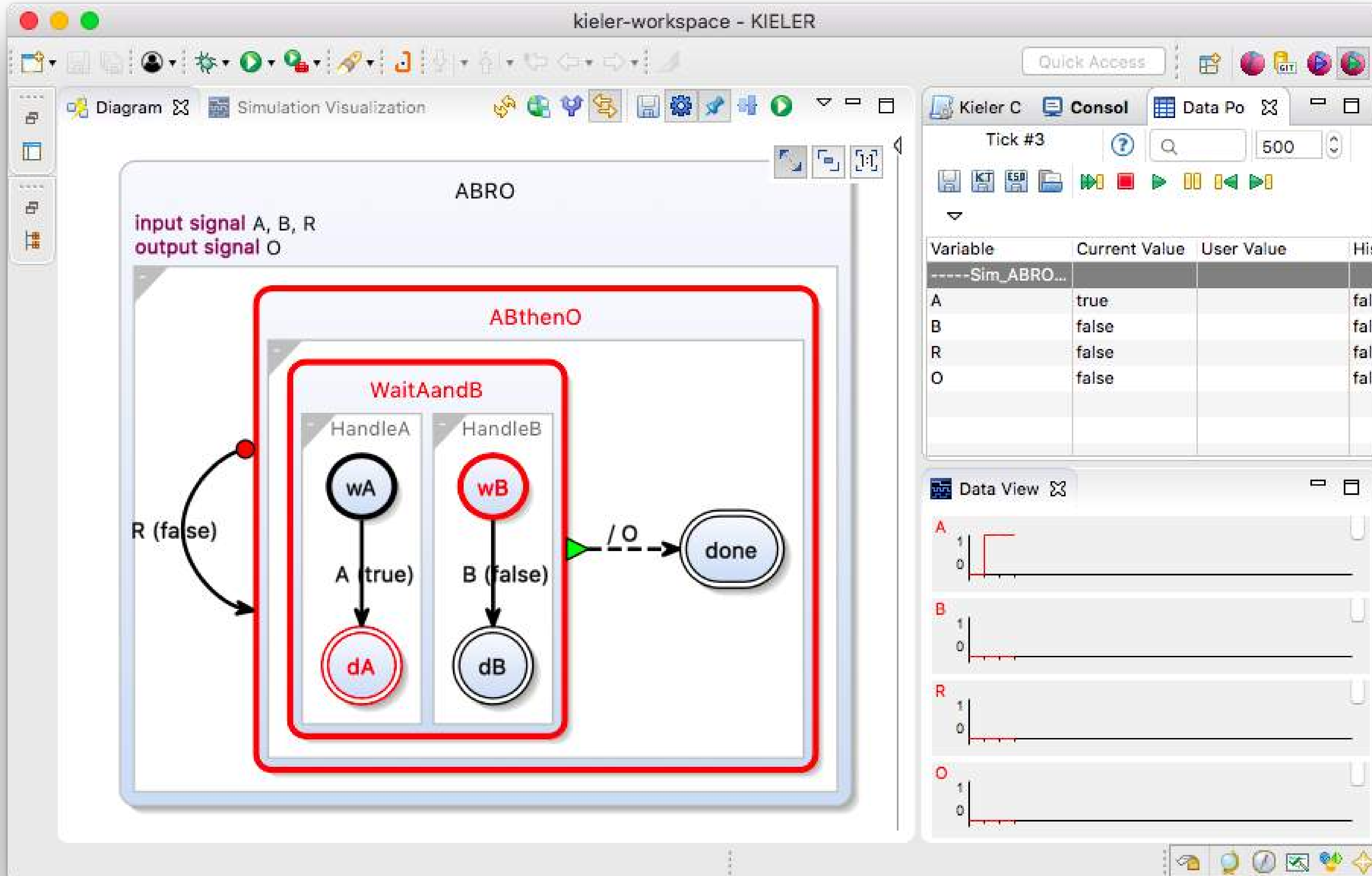
Simulation – Tick 1



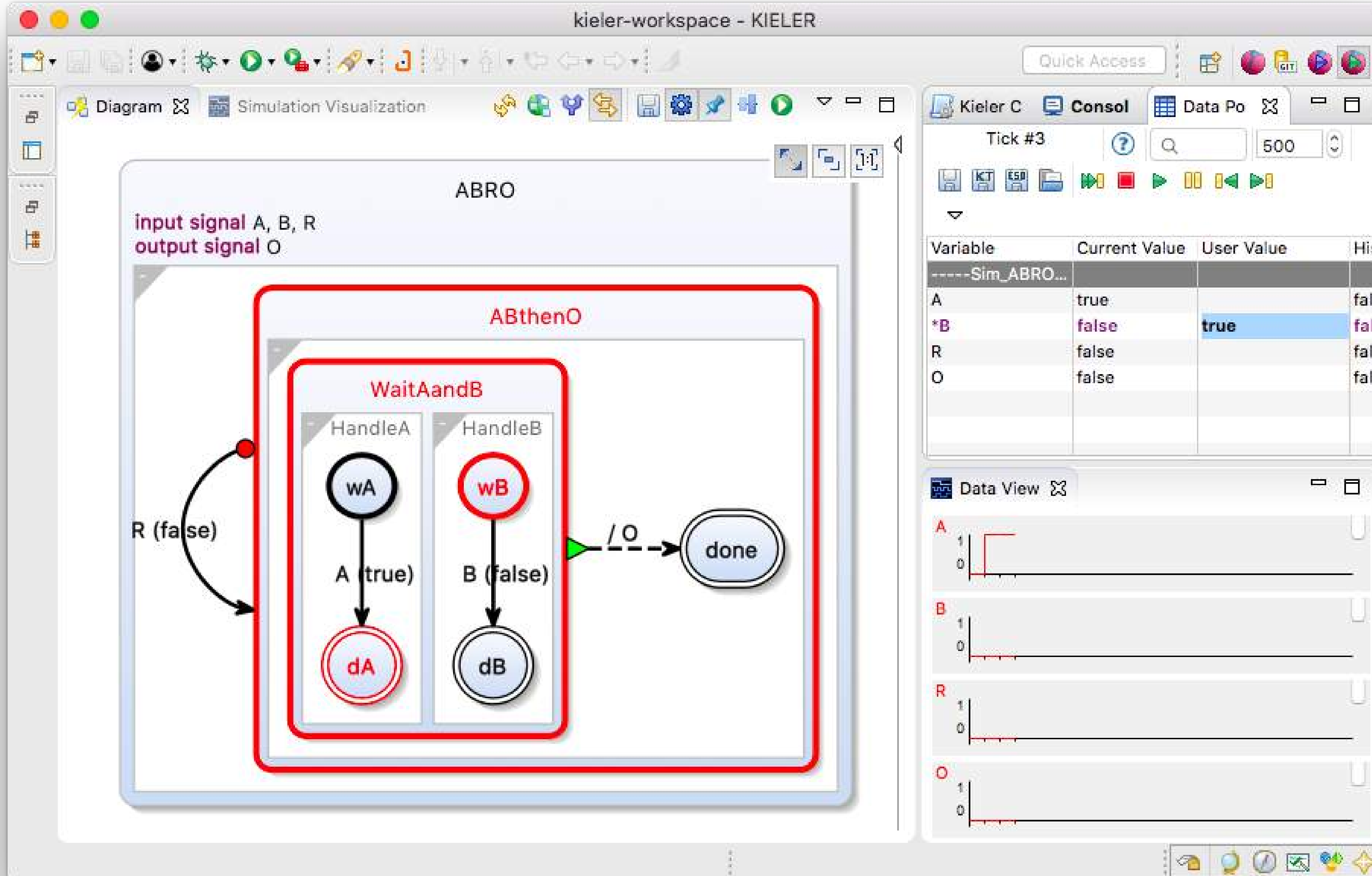
Simulation – Tick 2



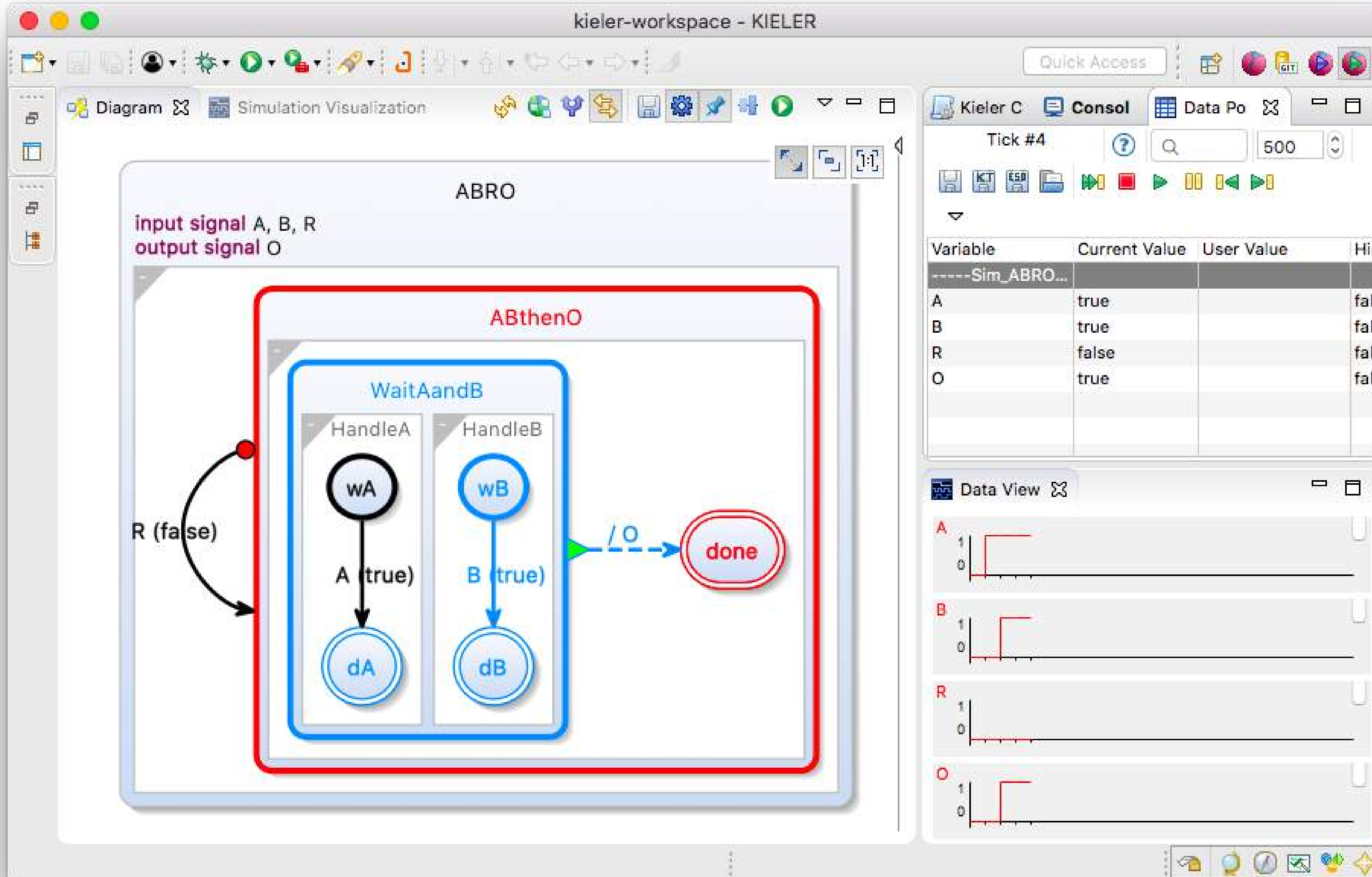
Simulation – Tick 3



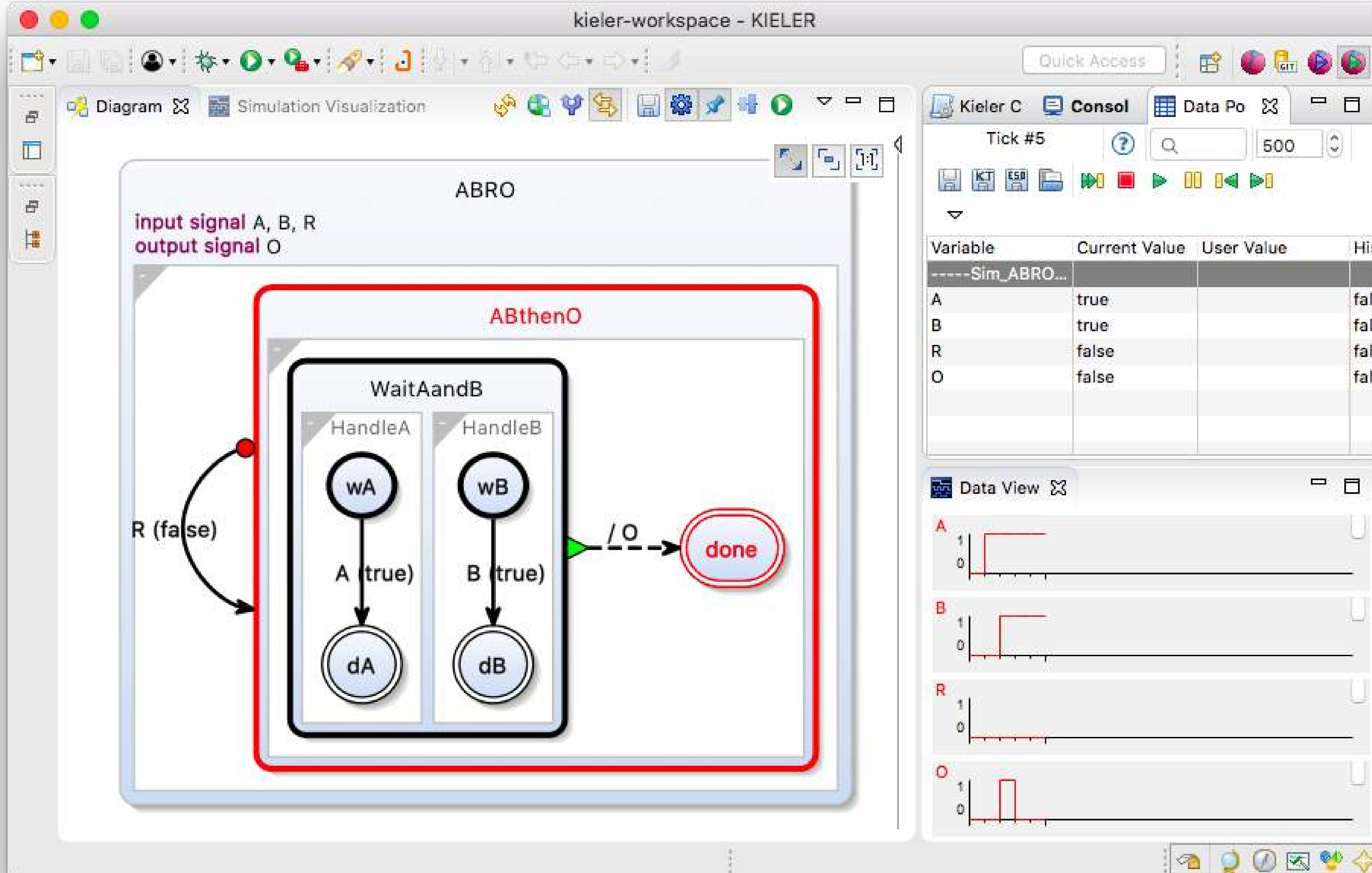
Simulation – Tick 3



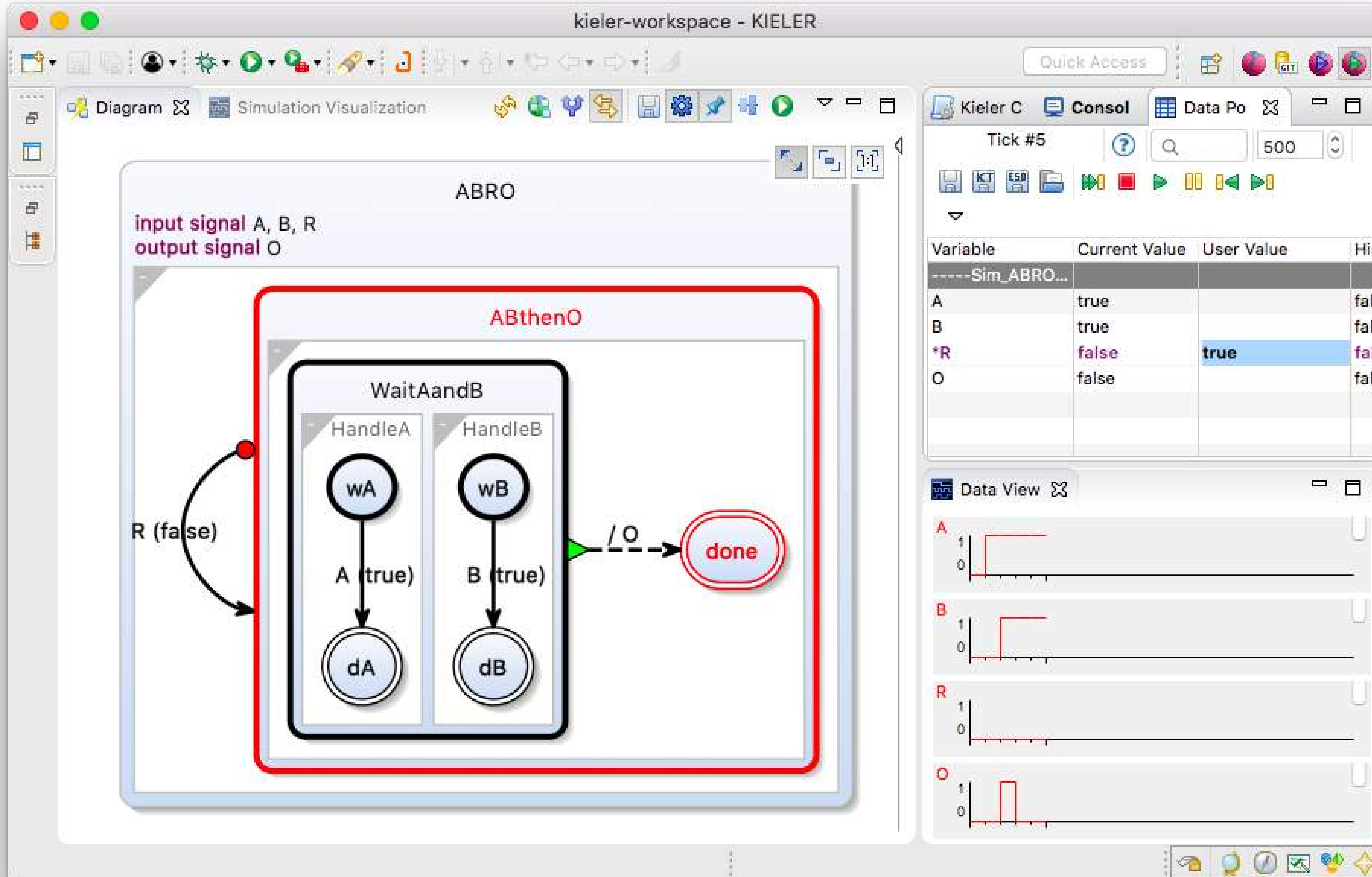
Simulation – Tick 4



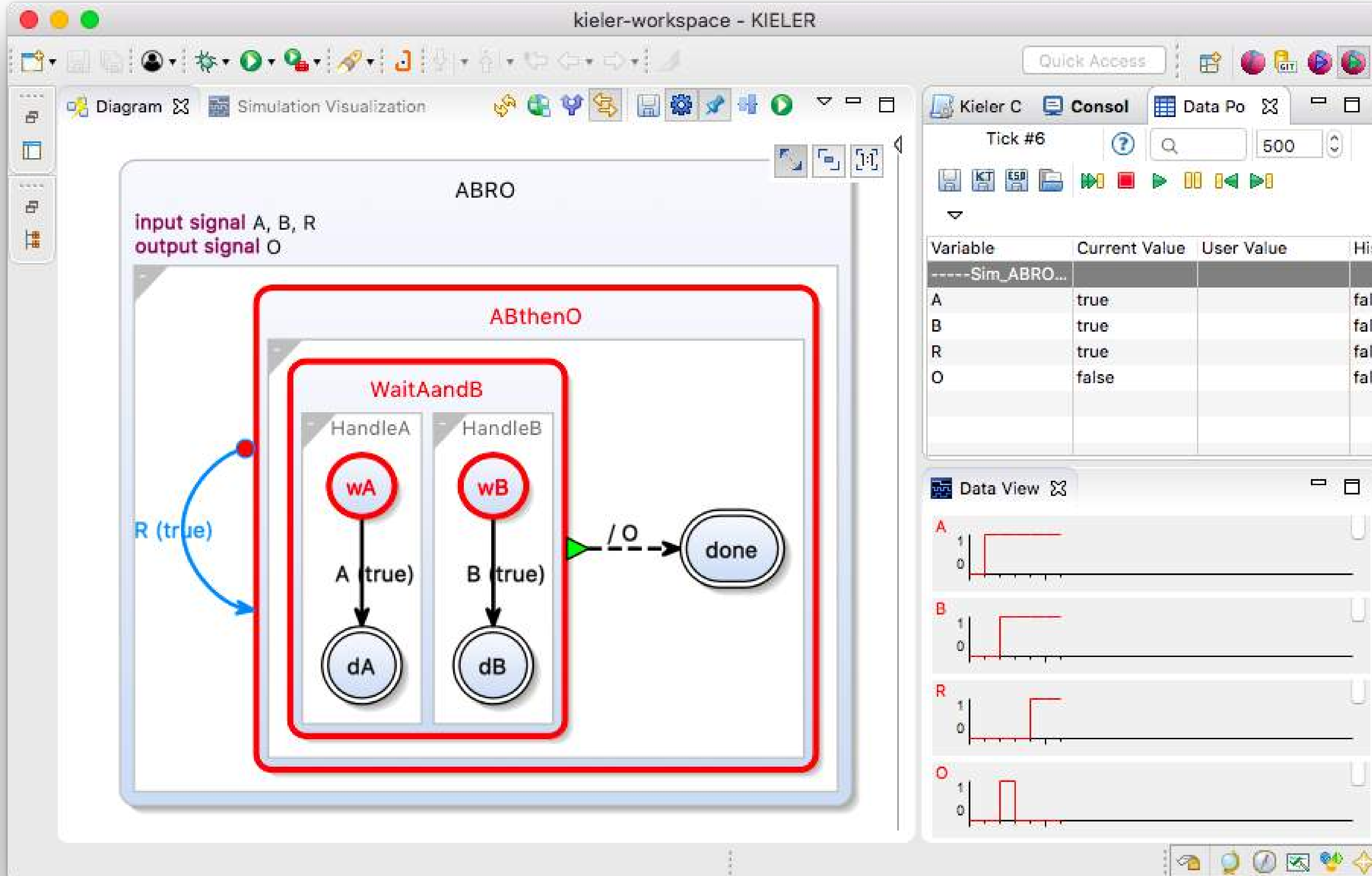
Simulation – Tick 5



Simulation – Tick 5

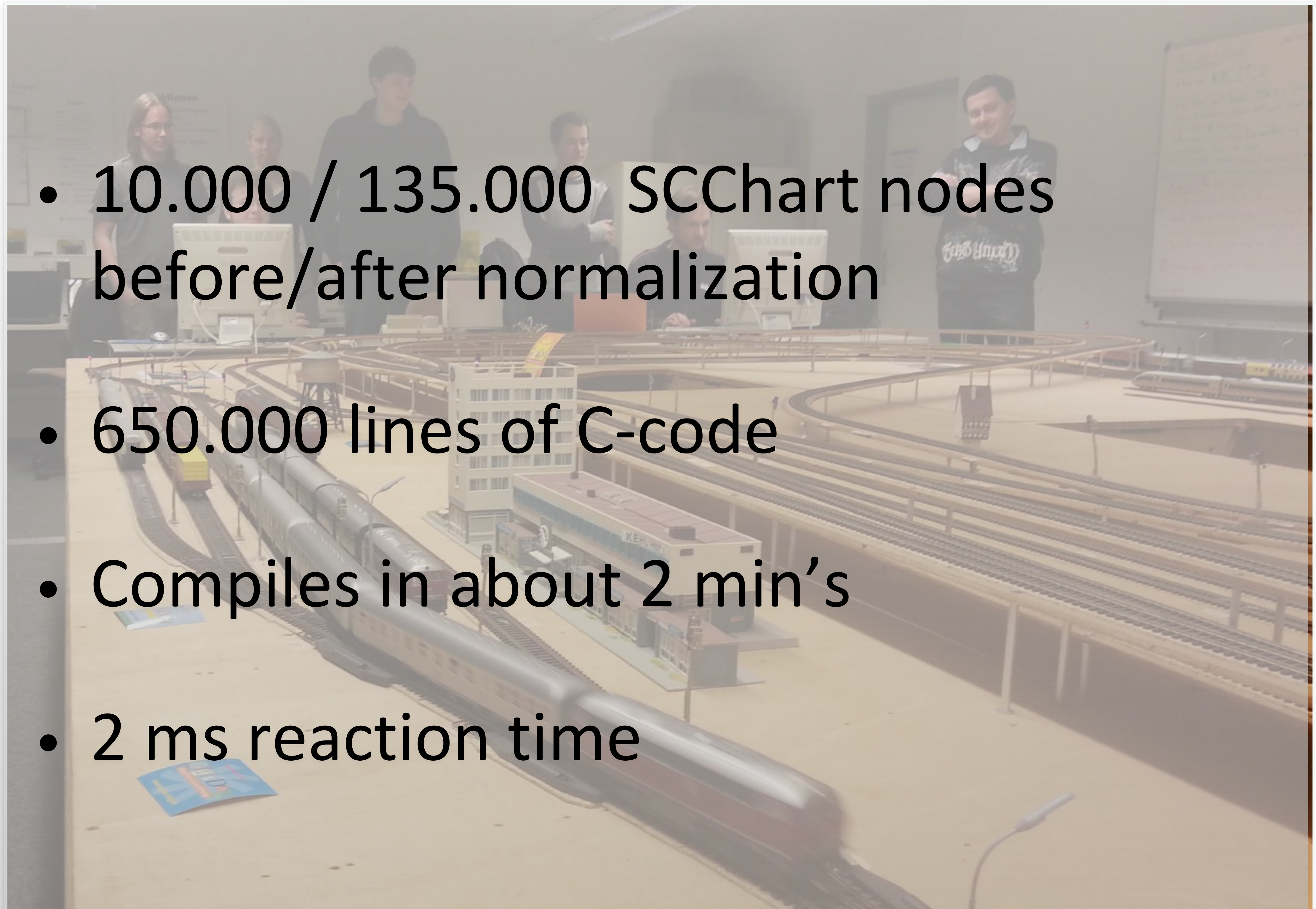


Simulation – Tick 6



SCCharts – Classroom-Tested

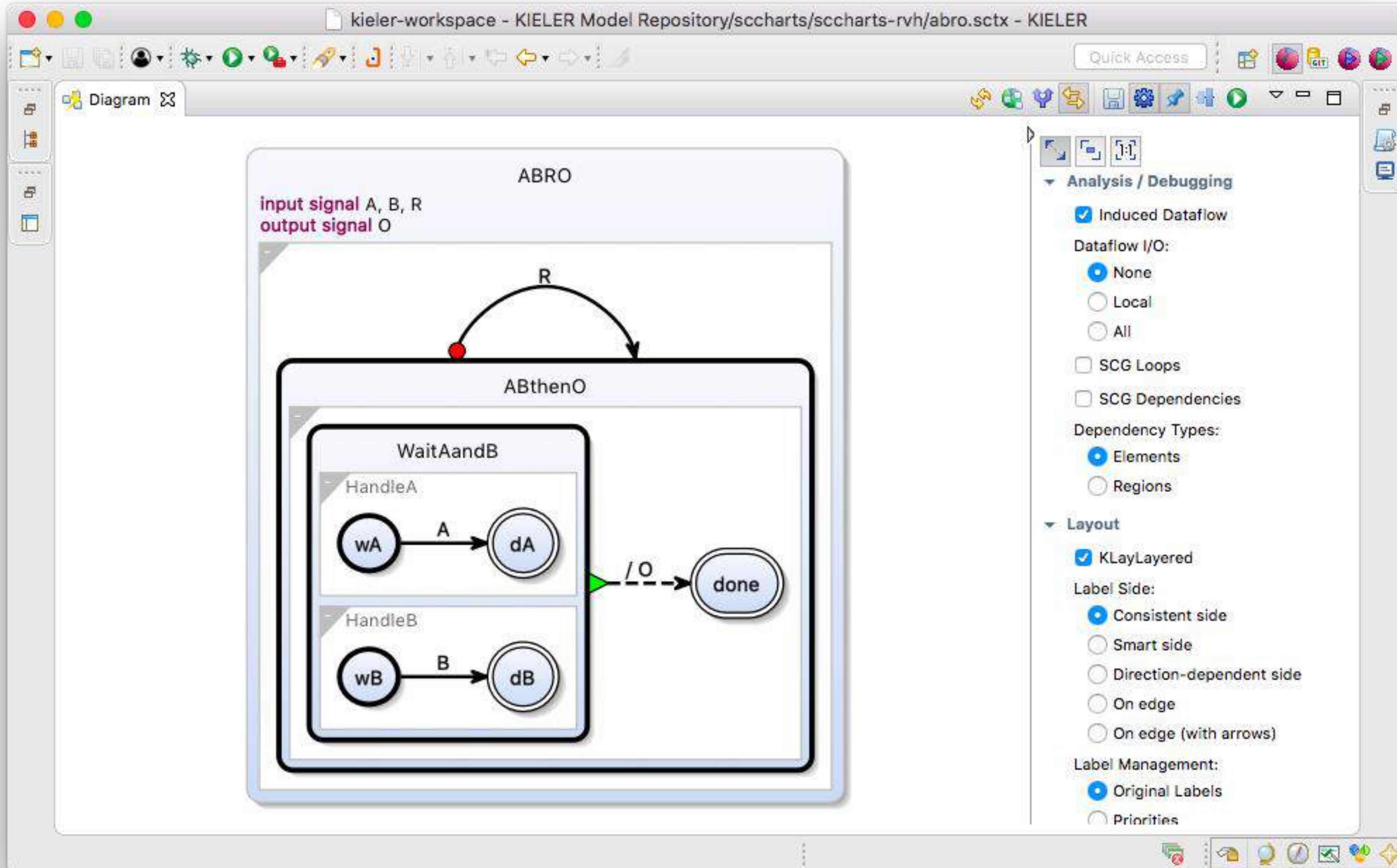
SCCharts – Classroom-Tested



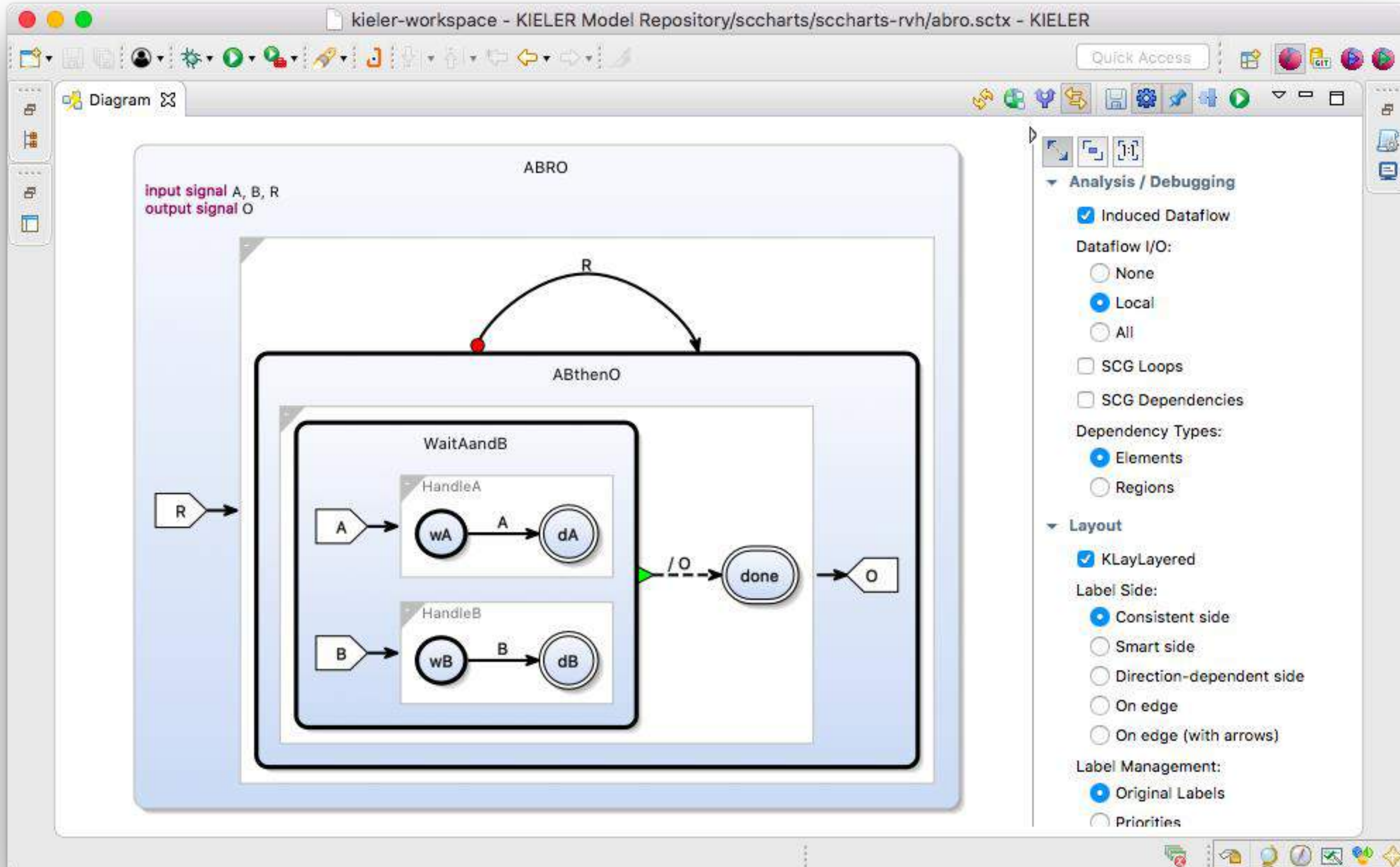
- 10.000 / 135.000 SCChart nodes before/after normalization
- 650.000 lines of C-code
- Compiles in about 2 min’s
- 2 ms reaction time

SCCharts – Modeling Dataflow

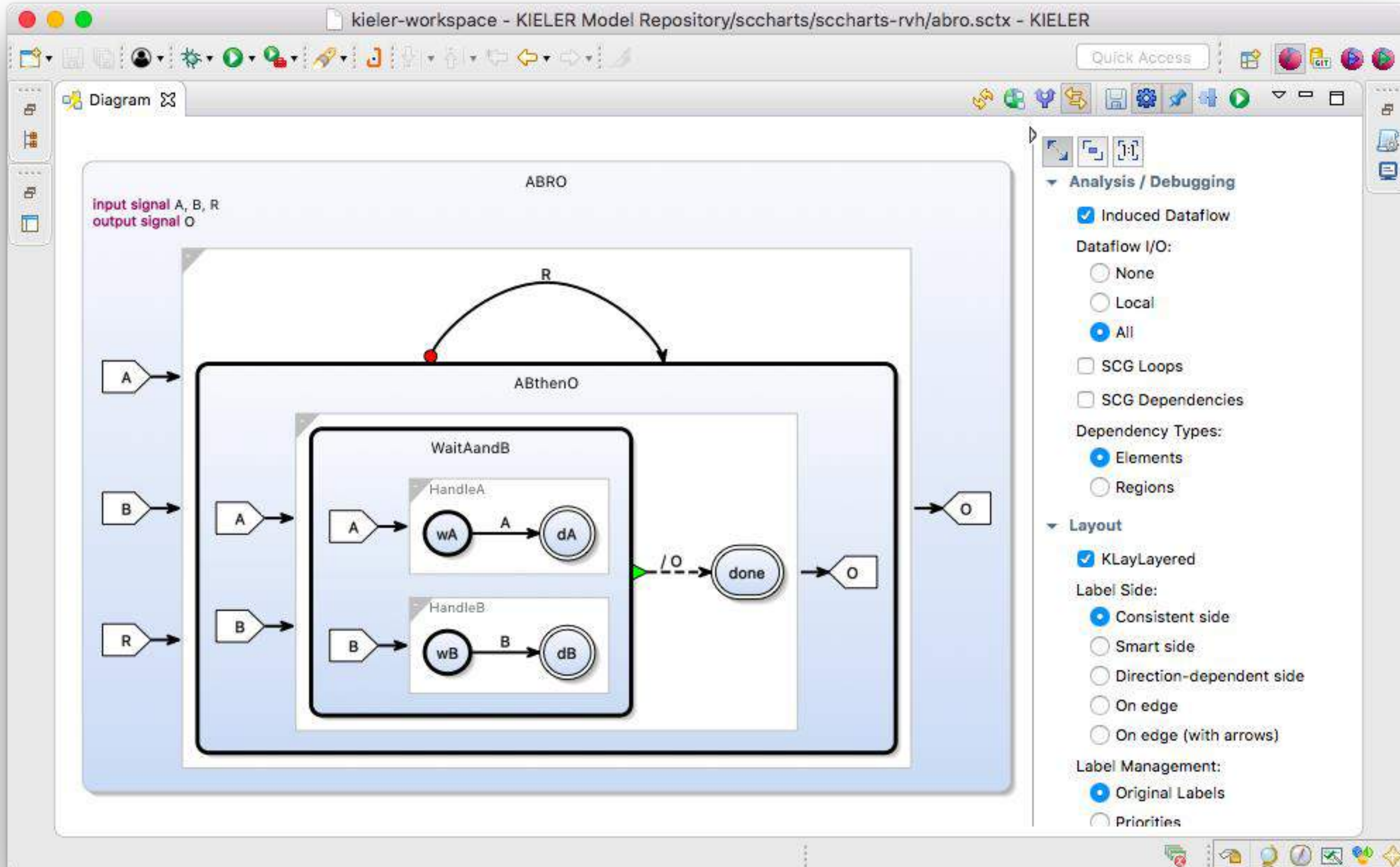
Dataflow View in ABRO



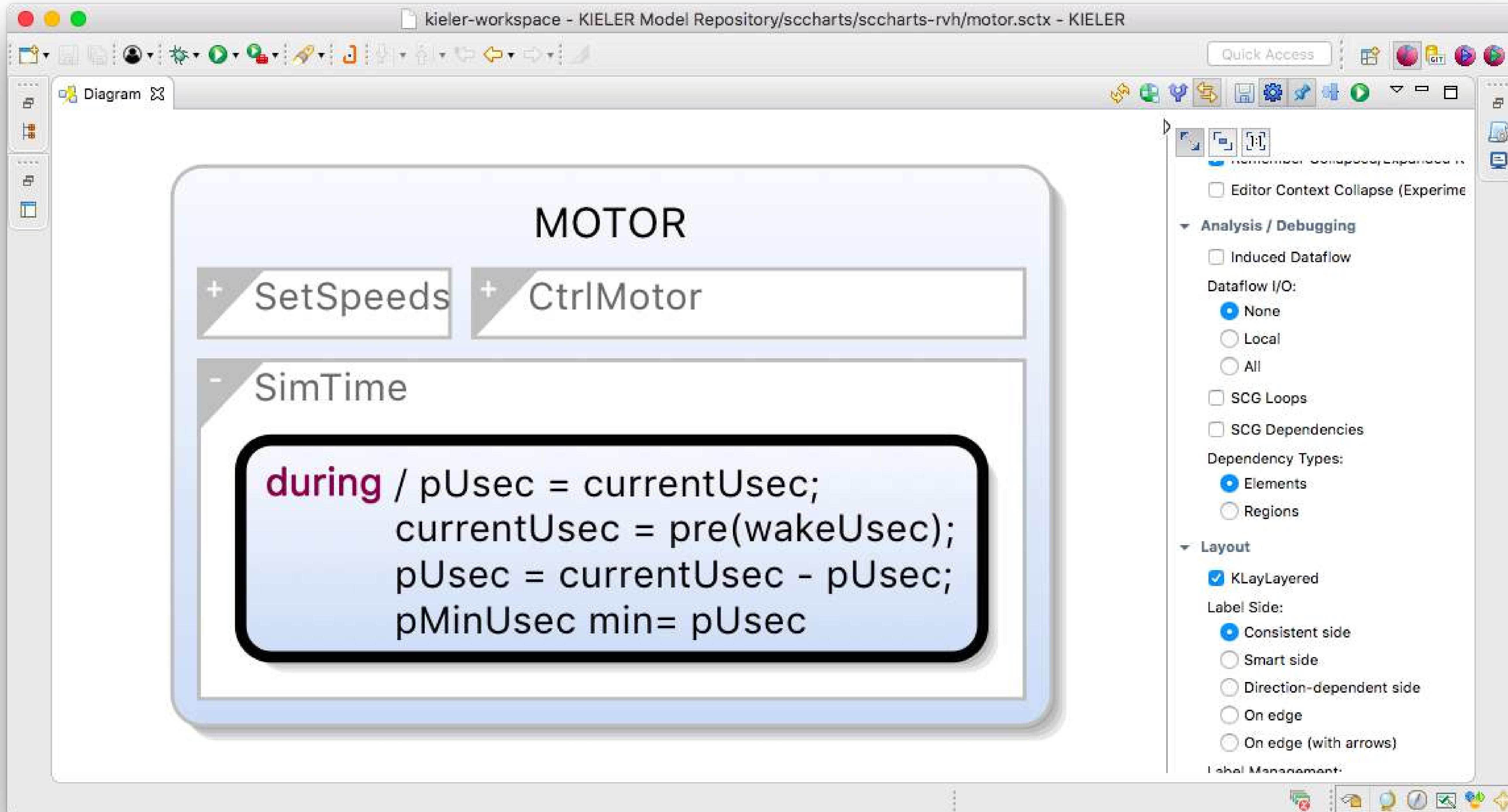
Dataflow View in ABRO



Dataflow View in ABRO

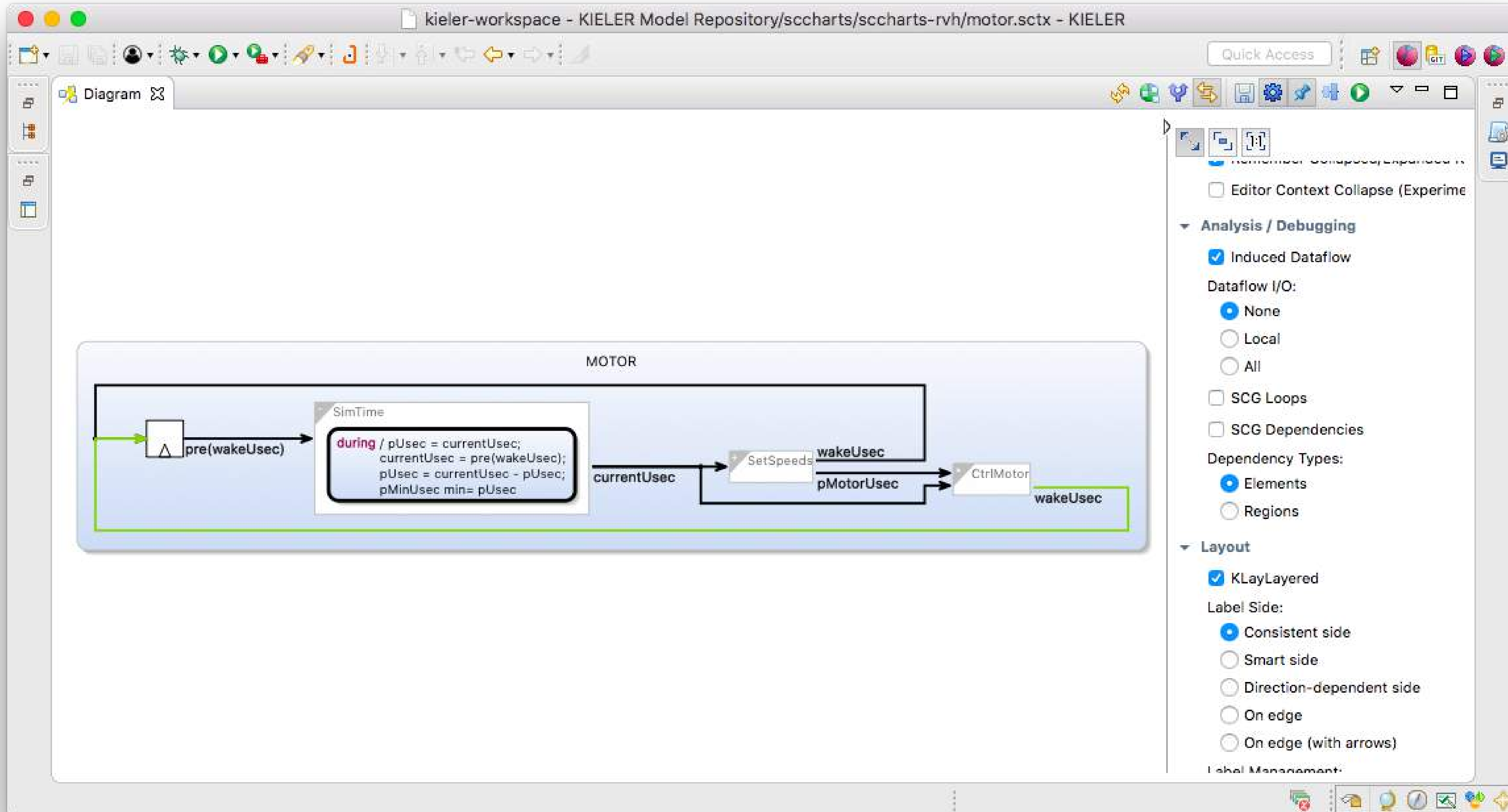


Dataflow View, with Communication



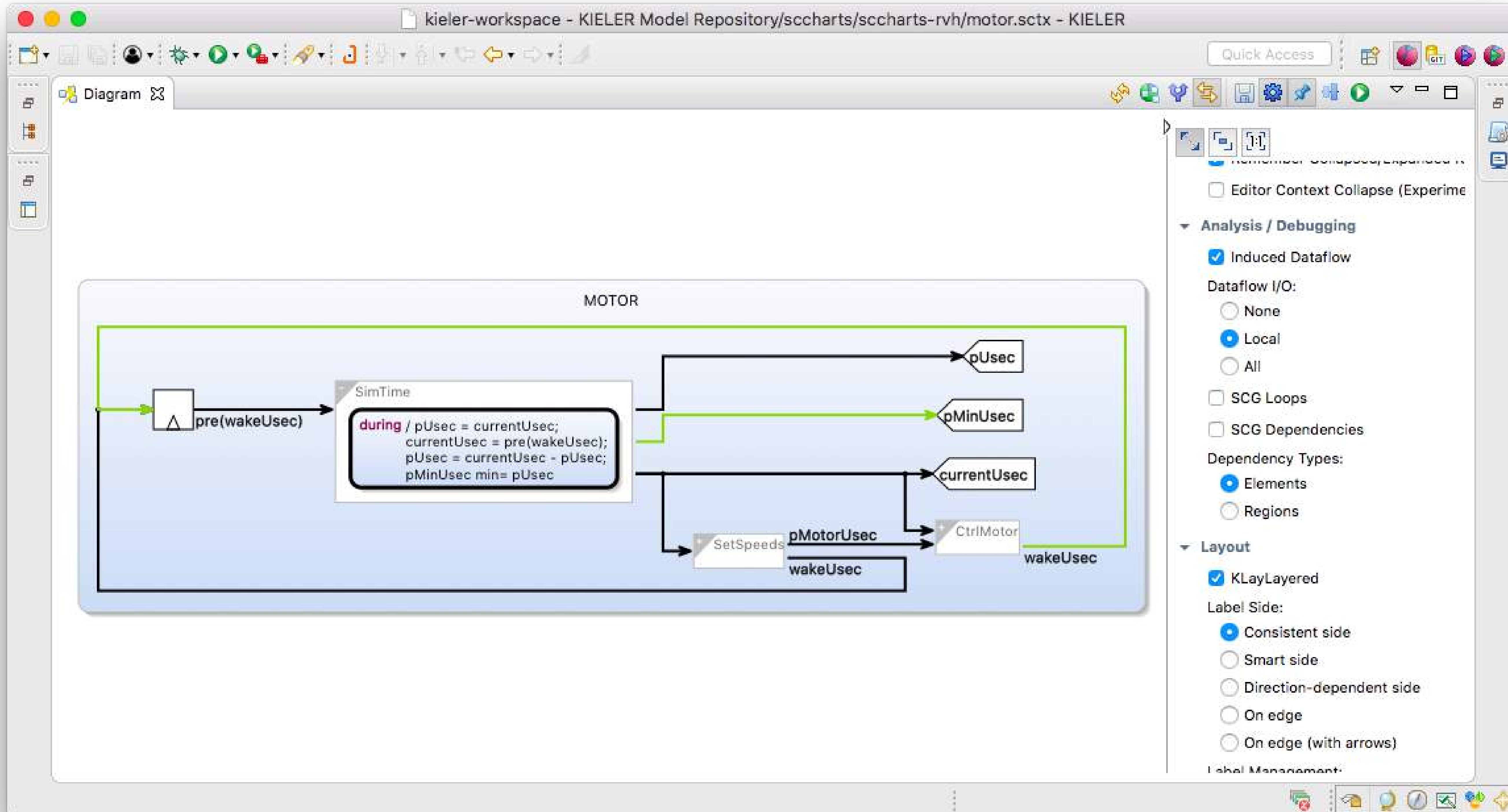
[Wechselberg, Schulz-Rosengarten, Smyth, von Hanxleden
Augmenting State Models with Data Flow
Principles of Modeling – LNCS Festschrift on Edward Lee's 60th Birthday (to appear)]

Dataflow View, with Communication



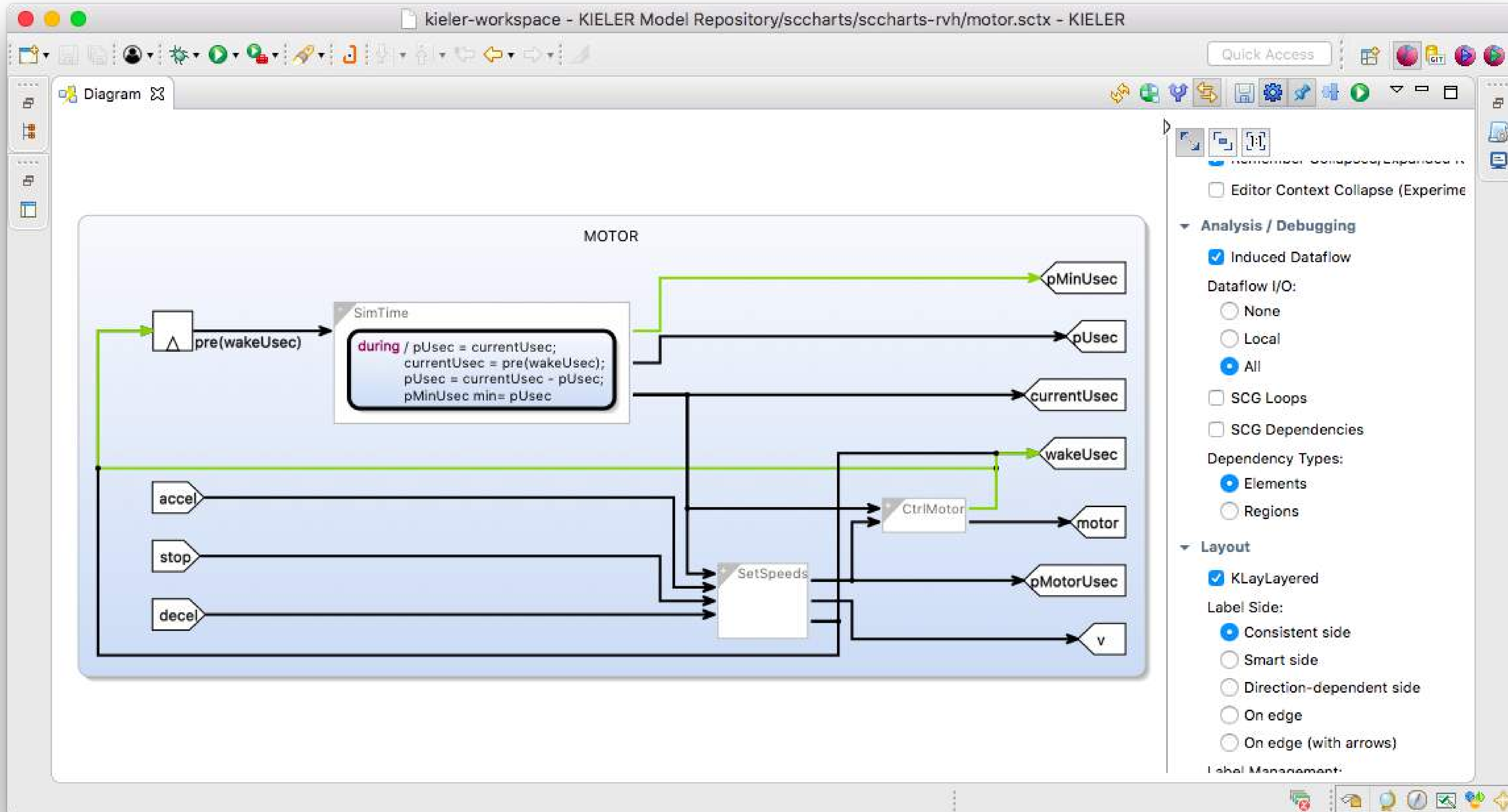
[Wechselberg, Schulz-Rosengarten, Smyth, von Hanxleden
Augmenting State Models with Data Flow
Principles of Modeling – LNCS Festschrift on Edward Lee's 60th Birthday (to appear)]

Dataflow View, with Communication

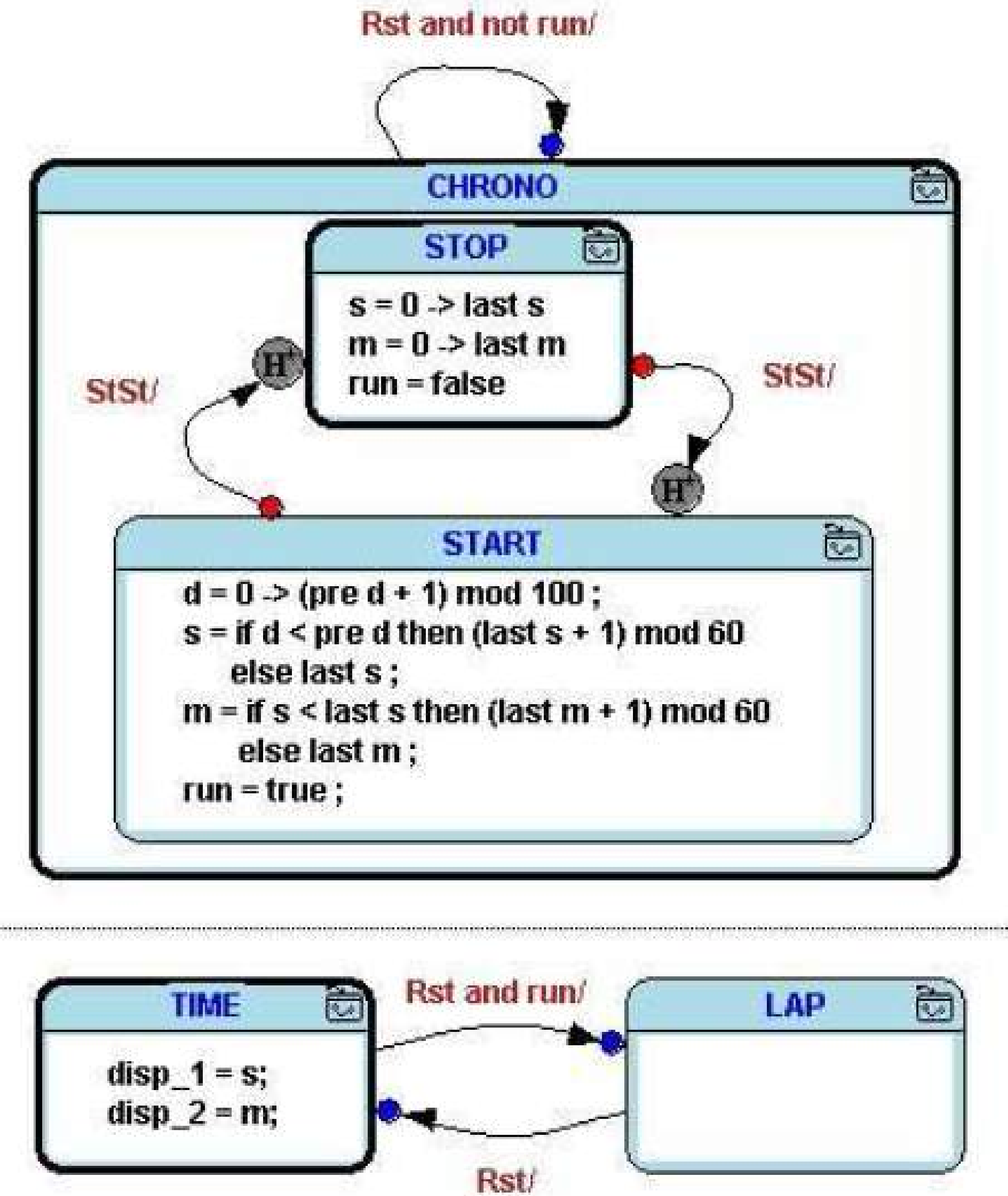


[Wechselberg, Schulz-Rosengarten, Smyth, von Hanxleden
Augmenting State Models with Data Flow
Principles of Modeling – LNCS Festschrift on Edward Lee's 60th Birthday (to appear)]

Dataflow View, with Communication



[Wechselberg, Schulz-Rosengarten, Smyth, von Hanxleden
Augmenting State Models with Data Flow
Principles of Modeling – LNCS Festschrift on Edward Lee's 60th Birthday (to appear)]




```
let node chrono (StSt, Rst) = (disp_1, disp_2) where
  automaton
    CHRONO ->
      do automaton
           STOP ->
             do  s = 0 -> last s
             and m = 0 -> last m
             and run = false
             unless StSt continue START
         | START ->
             let d = 0 -> (pre d + 1) mod 100 in
             do s = if d < pre d
                    then (last s + 1) mod 60
                    else last s
             and m = if s < last s
                     then (last m + 1) mod 60
                     else last m
             and run = true
             unless StSt continue STOP
         end
      until Rst and not run then CHRONO
  end and
  automaton
    TIME ->
      do disp_1 = s
      and disp_2 = m
      until Rst and run then LAP
  | LAP ->
      do until Rst then TIME
  end
```

Figure 2: A Chronometer

[Colaço, Pagano, Pouzet,
A Conservative Extension of Synchronous Dataflow with State Machines,
EMSOFT'05]

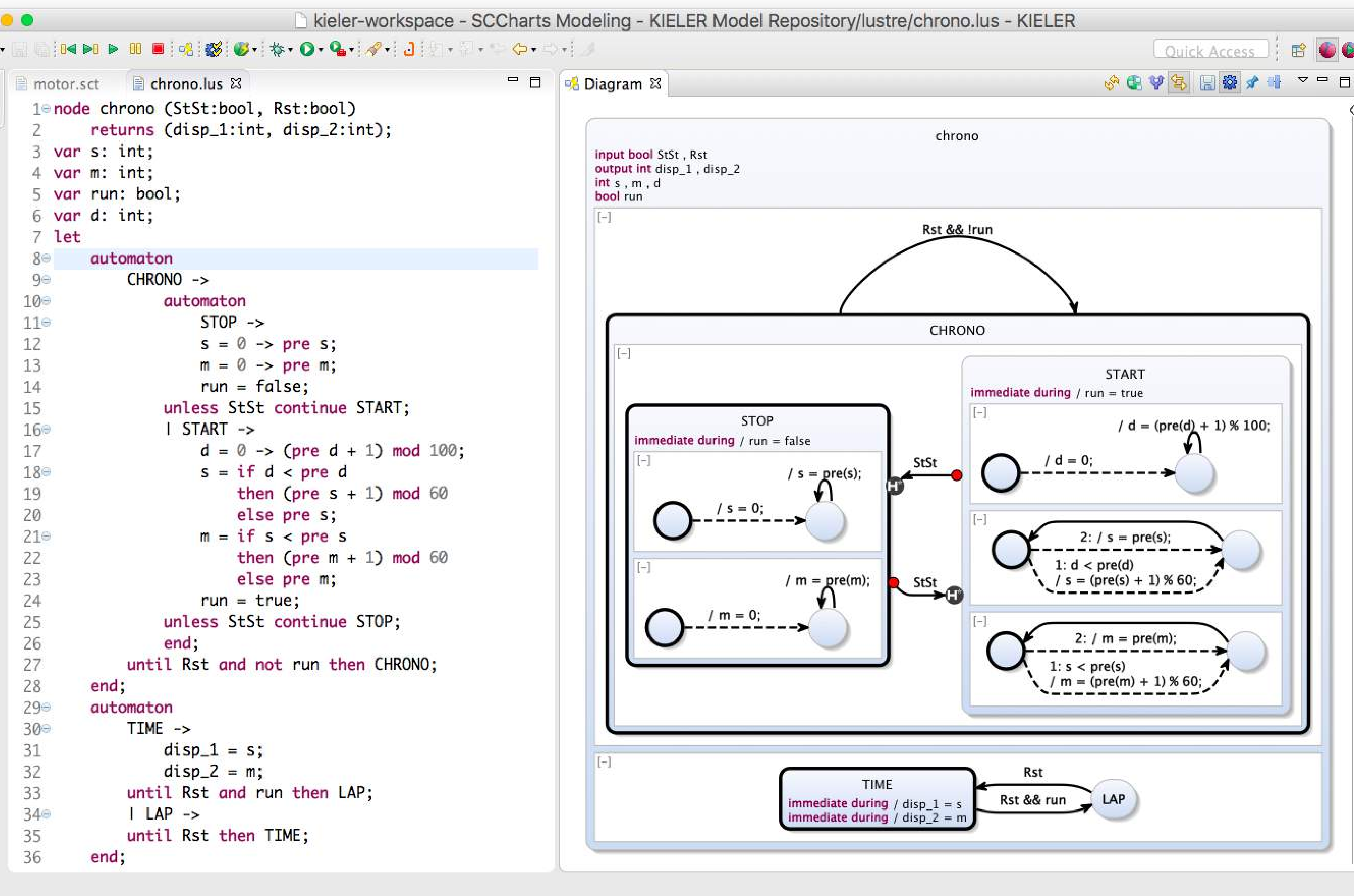
kieler-workspace - SCCharts Modeling - KIELER Model Repository/lustre/chrono.lus - KIELER

motor.sct | chrono.lus

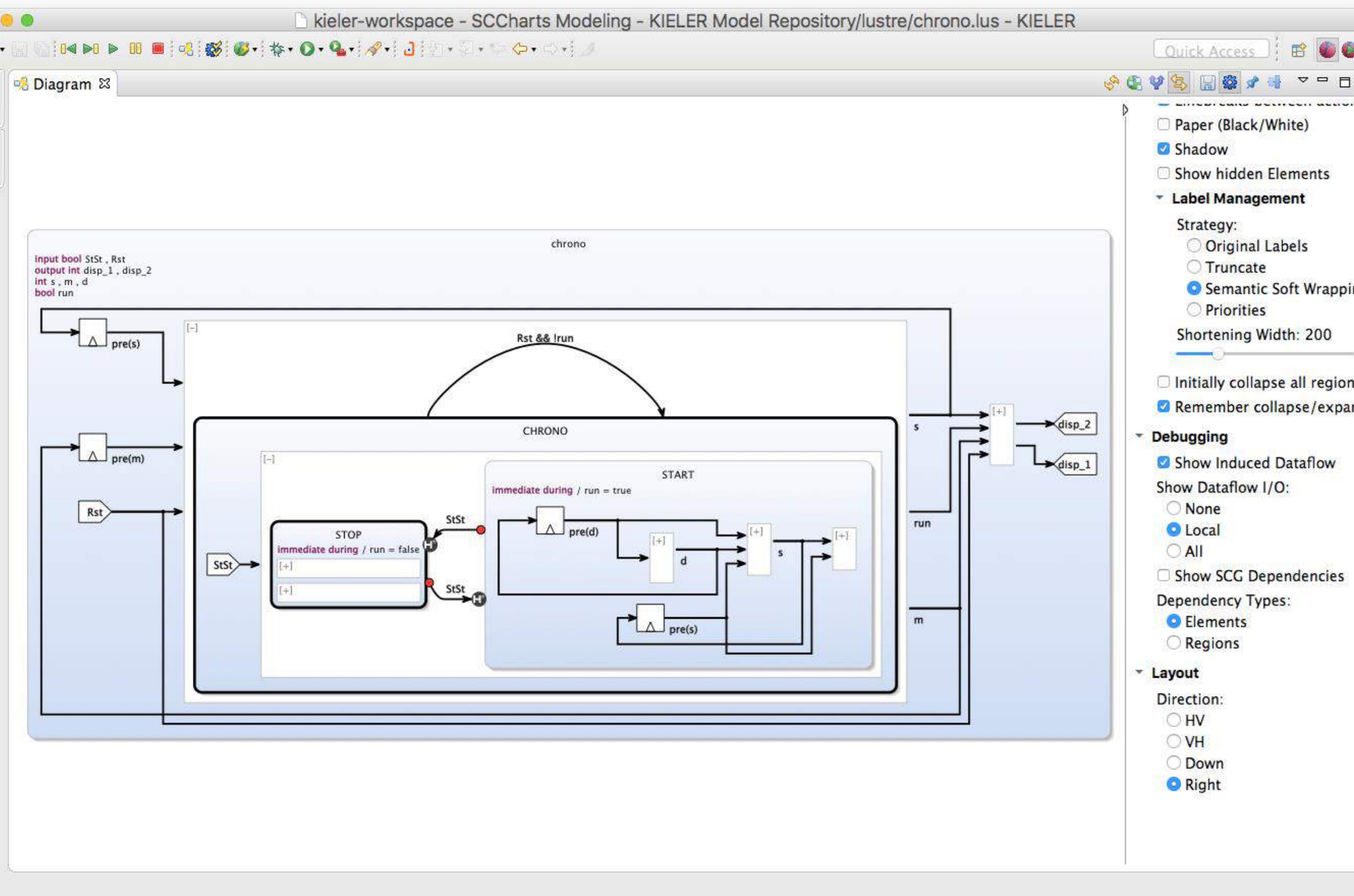
```
node chrono (StSt:bool, Rst:bool)
    returns (disp_1:int, disp_2:int);
var s: int;
var m: int;
var run: bool;
var d: int;
let
    automaton
        CHRONO ->
            automaton
                STOP ->
                s = 0 -> pre s;
                m = 0 -> pre m;
                run = false;
            unless StSt continue START;
            | START ->
                d = 0 -> (pre d + 1) mod 100;
                s = if d < pre d
                    then (pre s + 1) mod 60
                    else pre s;
                m = if s < pre s
                    then (pre m + 1) mod 60
                    else pre m;
                run = true;
            unless StSt continue STOP;
            end;
        until Rst and not run then CHRONO;
    end;
    automaton
        TIME ->
            disp_1 = s;
            disp_2 = m;
        until Rst and run then LAP;
        | LAP ->
        until Rst then TIME;
    end;
```

Writable | Insert | 8 : 9

[Clement Pascutto (ENS), Internship project at Kiel]



[Clement Pascutto (ENS), Internship project at Kiel]

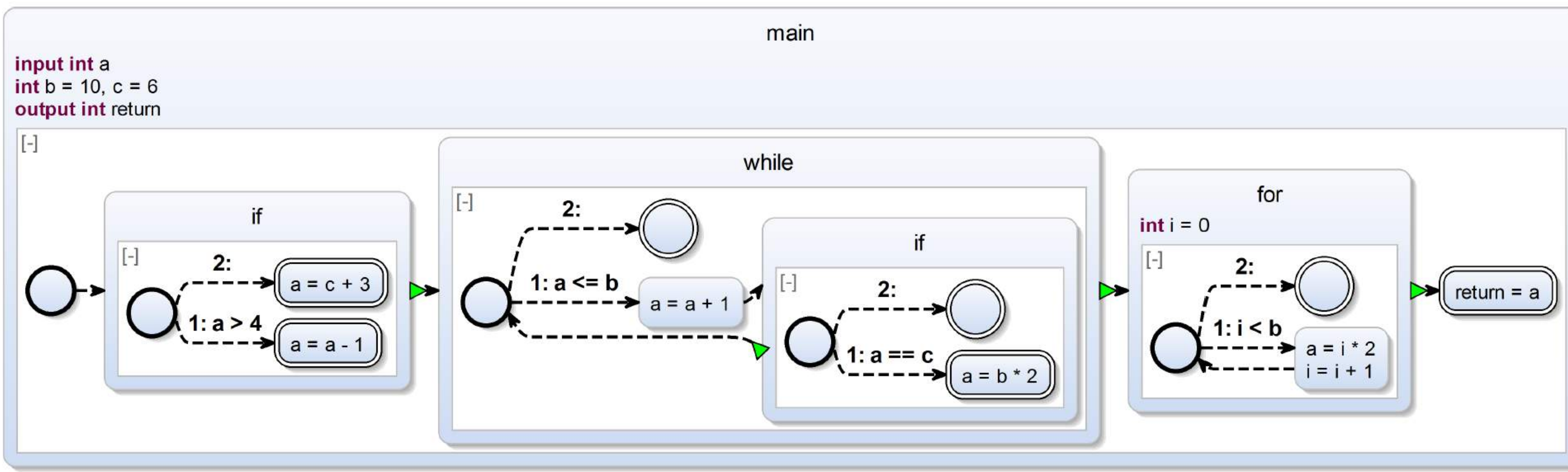


kieler-workspace - SCCharts Modeling - KIELER Model Repository/lustre/chrono.lus - KIELER
Quick Access
Diagram
chrono
input bool StSt , Rst
output int disp_1 , disp_2
int s , m , d
bool run
pre(s)
pre(m)
Rst
Rst && !run
CHRONO
STOP
immediate during / run = false
StSt
START
immediate during / run = true
pre(d)
d
s
pre(s)
run
m
disp_2
disp_1
Paper (Black/White)
Shadow
Show hidden Elements
Label Management
Strategy:
Original Labels
Truncate
Semantic Soft Wrappi
Priorities
Shortening Width: 200
Initially collapse all region
Remember collapse/expa
Debugging
Show Induced Dataflow
Show Dataflow I/O:
None
Local
All
Show SCG Dependencies
Dependency Types:
Elements
Regions
Layout
Direction:
HV
VH
Down
Right

From C to SCCharts

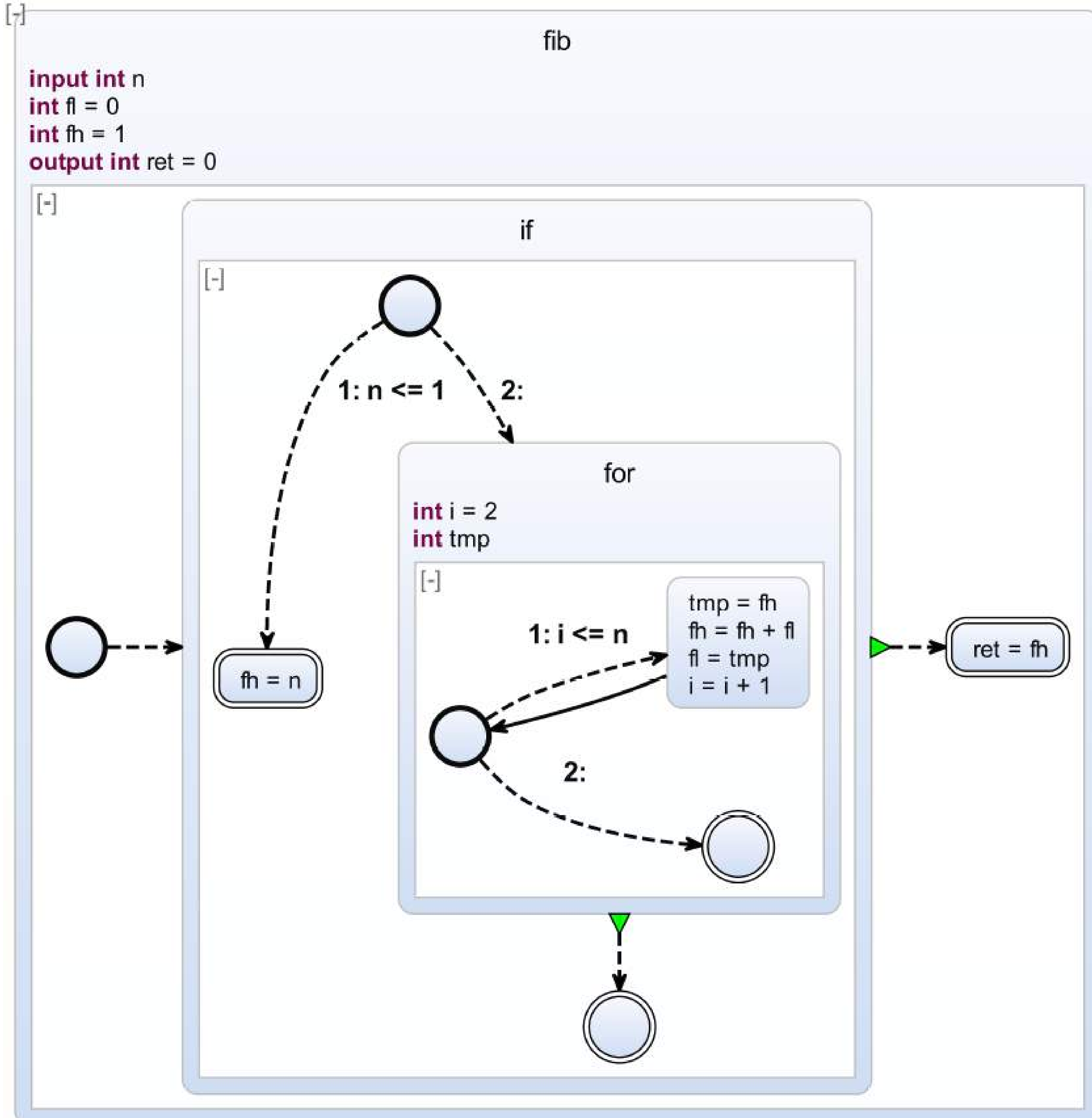
Extracting Visual Models from C

```
int main(int a) {
  int b = 10, c = 6;
  if (a > 4) {
    a = a − 1;
  } else {
    a = c + 3;
  }
  while (a <= b) {
    a = a + 1;
    if (a == c) {
      a = b ∗ 2;
    }
  }
  for (int i = 0; i < b; i = i + 1) {
    a = i ∗ 2;
  }
  return a;
}
```



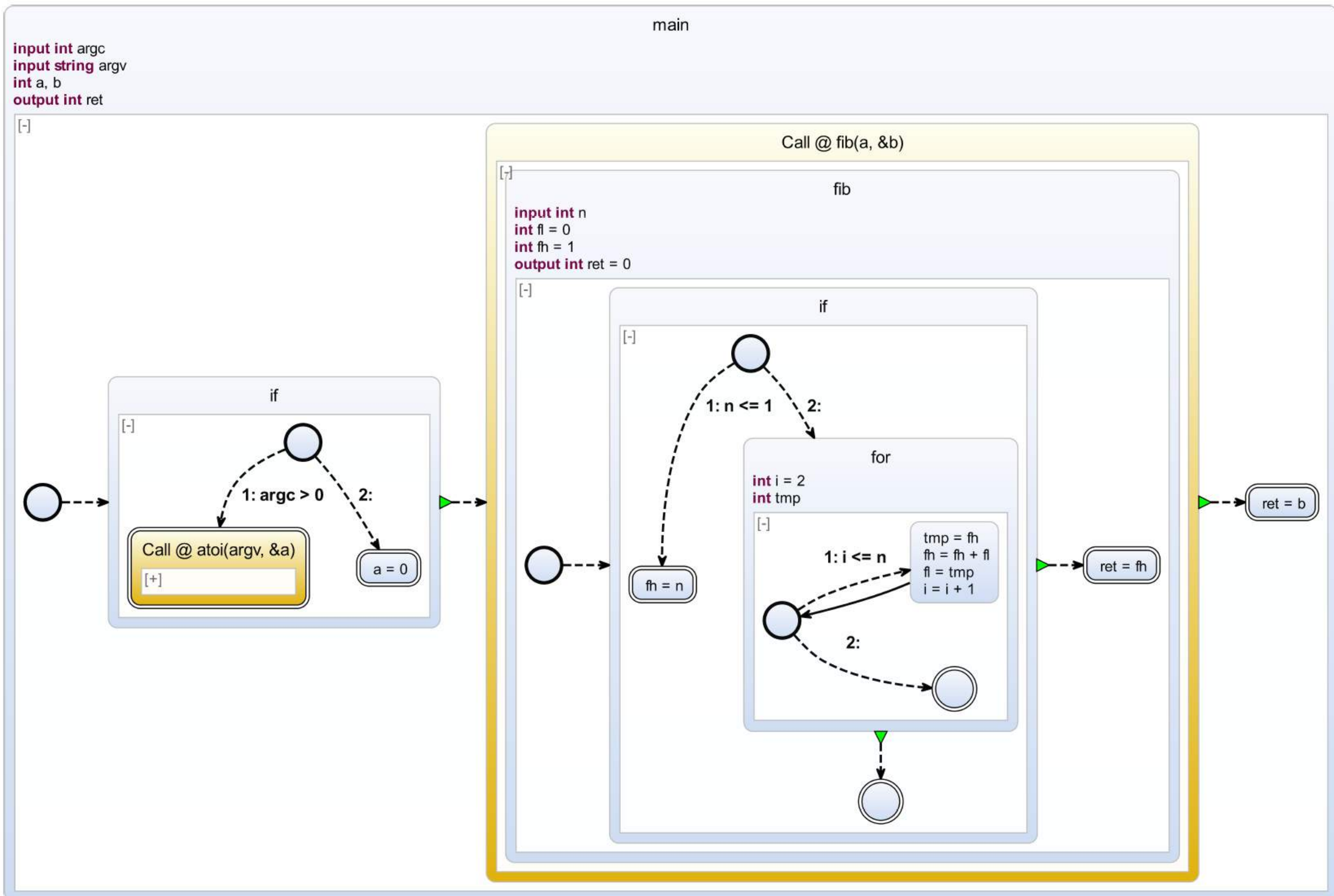
[Smyth, Lenga, von Hanxleden,
Model Extraction of Legacy C Code in SCCharts, ISoLA DS 2016]

```
int fib(int n) {
  int fl = 0, fh = 1;
  if (n<=1) { fh = n; }
  else {
    for (int i=2; i<=n; i++) {
      int tmp = fh;
      fh += fl;
      fl = tmp;
  }}
  return fh;
}
```



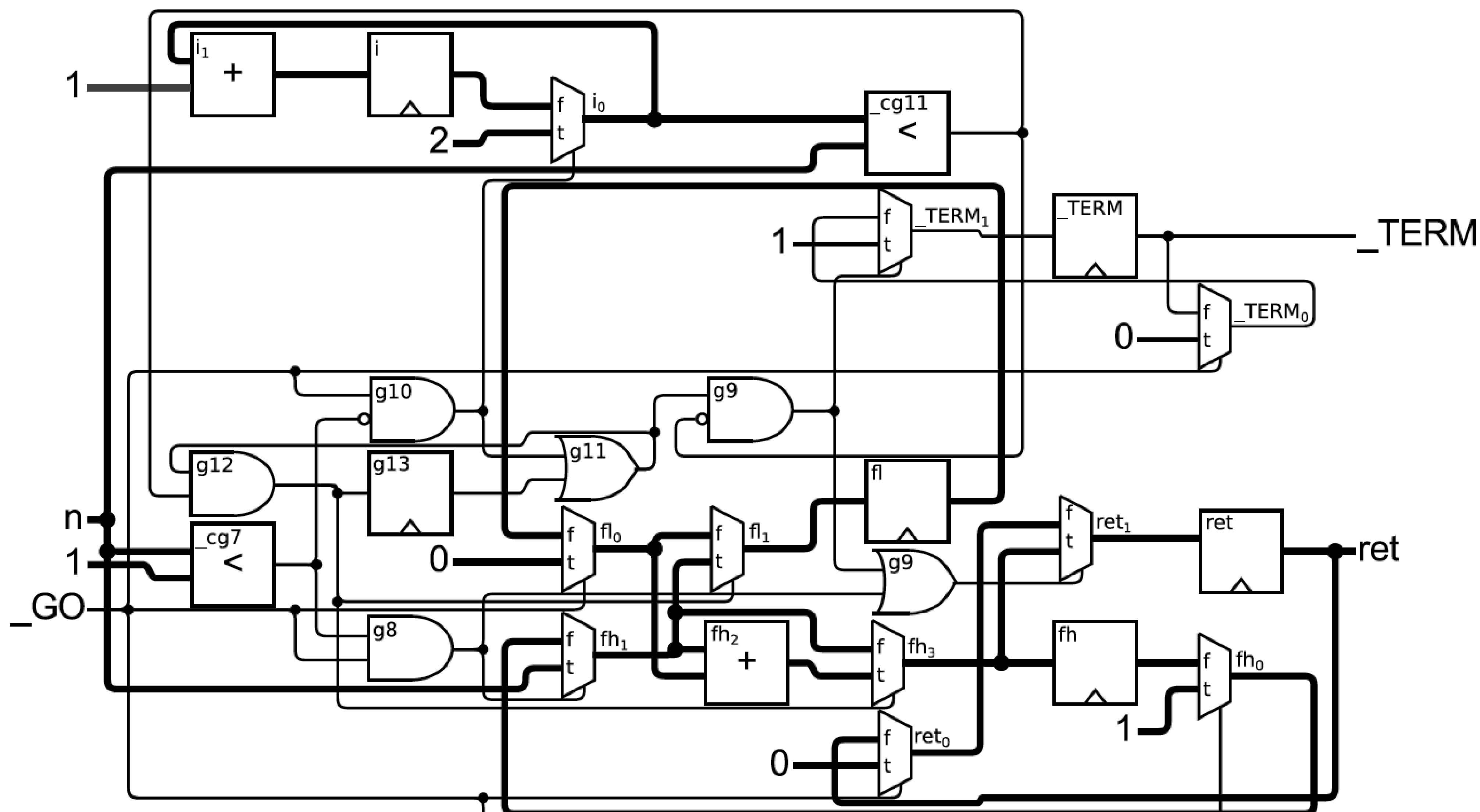
```
int main(int argc, char** argv) {
  int a, b;
  if (argc>0) {
    a = atoi(argv[0]);
  } else {
    a = 0;
  }
  b = fib(a);
  return b;
}
```

```
int fib(int n) {
  int fl = 0, fh = 1;
  if (n<=1) { fh = n; }
  else {
    for (int i=2; i<=n; i++) {
      int tmp = fh;
      fh += fl;
      fl = tmp;
```



```
typedef struct {
  char _GO;
  char g7;
  ...
} TickData1;

void reset1(TickData1 *d) {
  d->pg12 = 0;
  d->_GO = 1;
  d->_TERM = 0;
}

void tick1(TickData1 *d) {
  tickLogic1(d);
  d->_GO = 0;
  d->pg12 = d->g12;
}
void tickLogic1(TickData1 *d) {
  d->g7 = d->_GO;
  if (d->g7) {
    d->fl = 0;
    d->fh = 1;
  }
  d->_cg7 = d->n <= 1;
  d->g8 = d->g7 && d->_cg7;
  if (d->g8) {
    d->fh = d->n;
  }
  d->g13 = d->pg12;
  d->g10 = d->g7 && !d->_cg7;
  if (d->g10) {
    d->__fib_int_local_i = 2;
  }
  d->g11 = d->g13 || d->g10;
  d->_cg11 =
    d->__fib_int_local_i <= d->n;
  d->g12 = d->g11 && d->_cg11;
  if (d->g12) {
    d->__fib_int_local_tmp = d->fh;
    d->fh = d->fh + d->fl;
    d->fl = d->__fib_int_local_tmp;
    d->__fib_int_local_i =
      d->__fib_int_local_i + 1;
  }
  d->g9 = d->g11 &&
   !d->_cg11 || d->g8;
  if (d->g9) {
    d->ret = d->fh;
    d->_TERM = 1;
  }
}
```

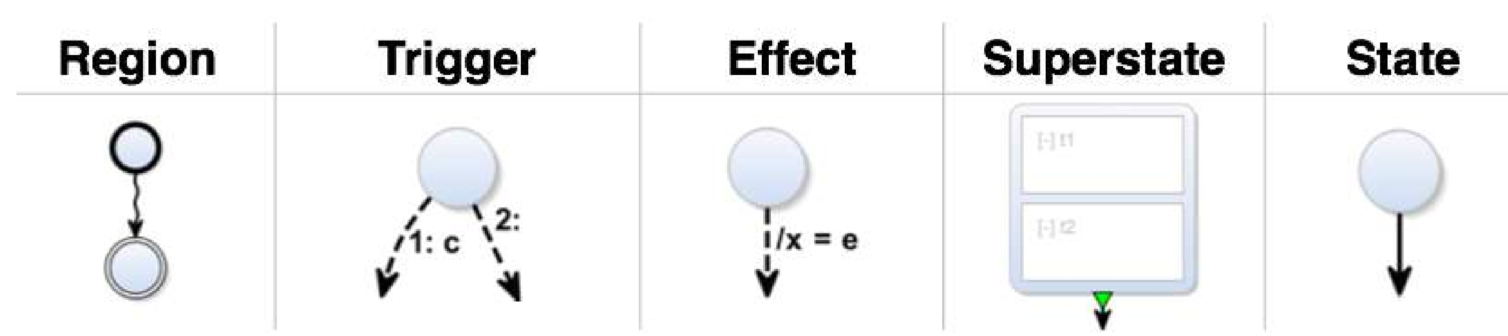
```c
typedef struct {
  char _GO;
  char g7;
  ...
} TickData1;

void reset1(TickData1 *d) {
  d->pg12 = 0;
  d->_GO = 1;
  d->_TERM = 0;
}

void tick1(TickData1 *d) {
  tickLogic1(d);
  d->_GO = 0;
  d->pg12 = d->g12;
}
void tickLogic1(TickData1 *d) {
  d->g7 = d->_GO;
  if (d->g7) {
    d->fl = 0;
    d->fh = 1;
  }
  d->_cg7 = d->n <= 1;
  d->g8 = d->g7 && d->_cg7;
  if (d->g8) {
    d->fh = d->n;
  }
  d->g13 = d->pg12;
  d->g10 = d->g7 && !d->_cg7;
  if (d->g10) {
    d->__fib_int_local_i = 2;
  }
  d->g11 = d->g13 || d->g10;
  d->_cg11 =
    d->__fib_int_local_i <= d->n;
  d->g12 = d->g11 && d->_cg11;
  if (d->g12) {
    d->__fib_int_local_tmp = d->fh;
    d->fh = d->fh + d->fl;
    d->fl = d->__fib_int_local_tmp;
    d->__fib_int_local_i =
      d->__fib_int_local_i + 1;
  }
  d->g9 = d->g11 &&
   !d->_cg11 || d->g8;
  if (d->g9) {
    d->ret = d->fh;
    d->_TERM = 1;
  }
}
```



SCCharts Wrap-Up

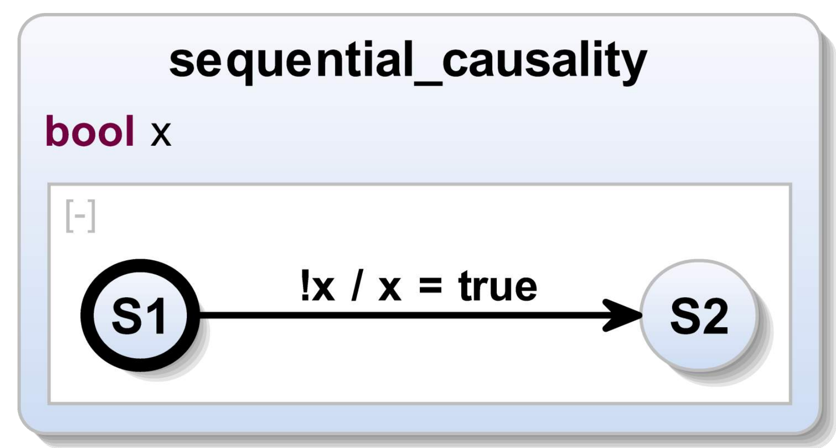
Language

- 5 core constructs
- Smörgåsboard of extensions



Model of Computation

- Relaxed synchrony
- Still determinate
- Can model C programs



Compilation

- M2M transformations
- Stress-tested in KIELER

KIELER
The Key to Efficient Modeling

Still plenty of things to do: Variants on SC MoC, optimize code generation, pragmatics improvements for schedulability analysis, …

Code Generation

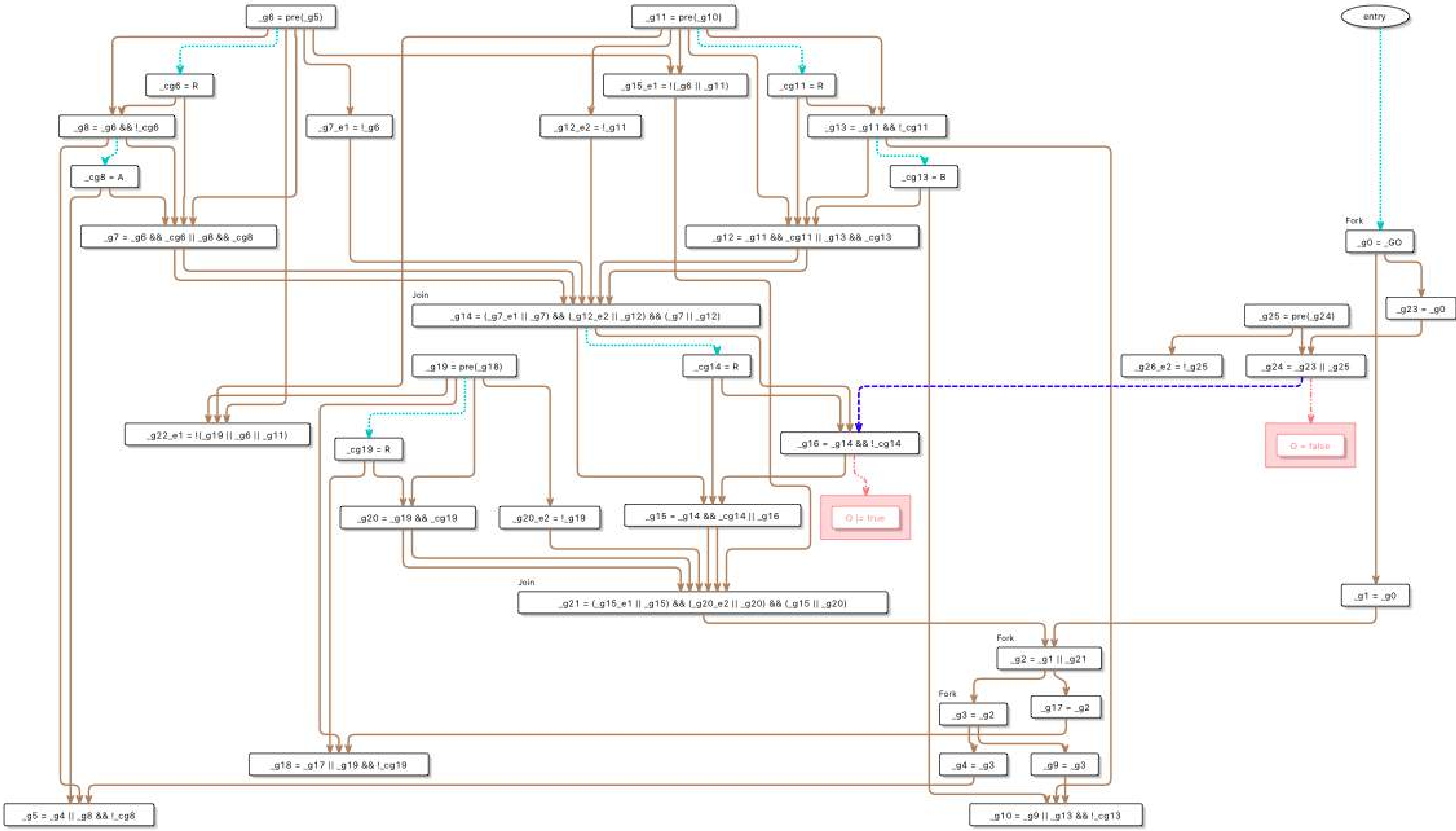
Downstream Compilation

So far, two alternative compilation strategies from SCL/SCG to C/VHDL

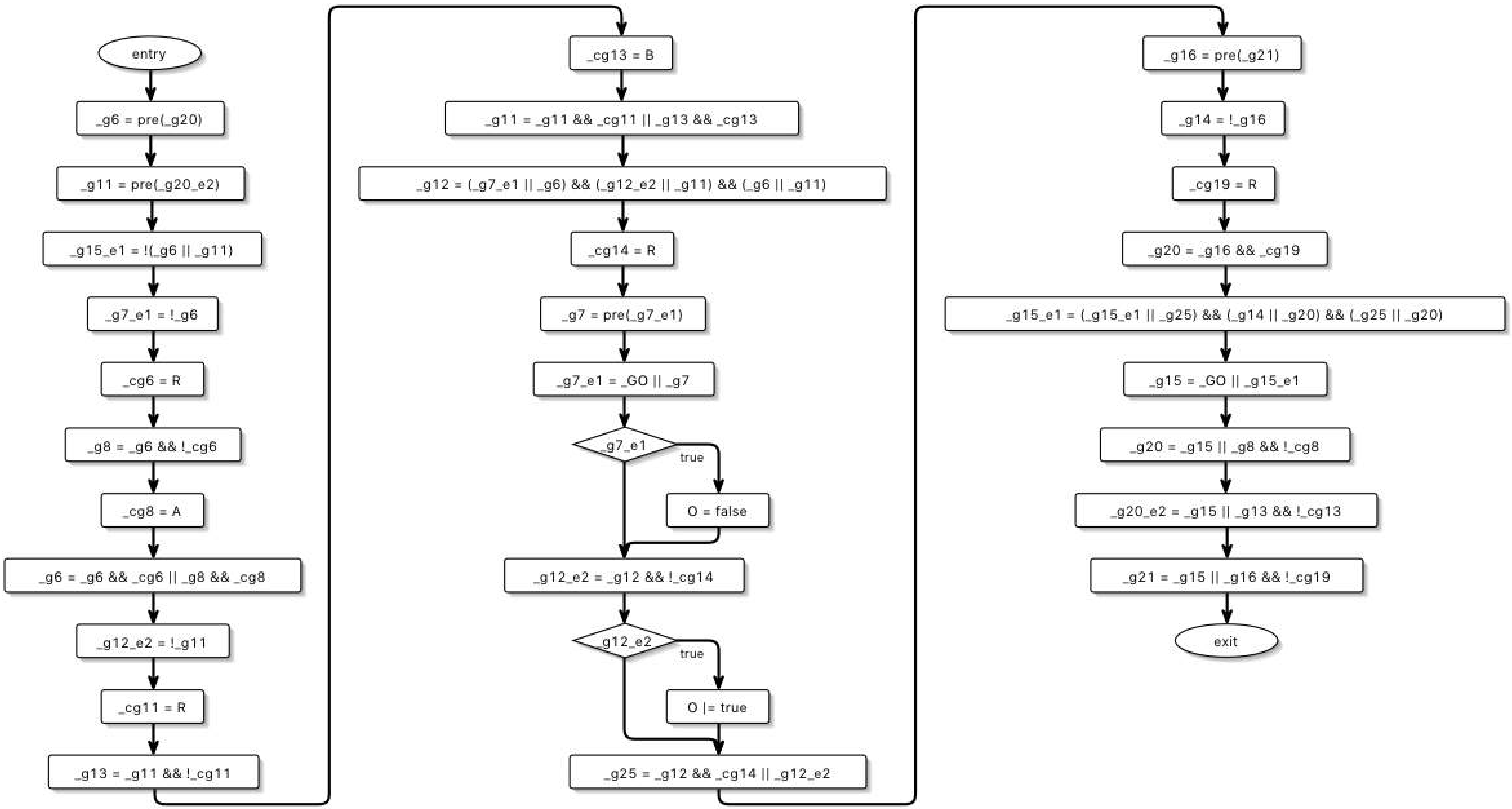
	Dataflow	Priority
Accepts instantaneous loops	–	+
Can synthesize hardware	+	–
Can synthesize software	+	+
Size scales well (linear in size of SCChart)	+	+
Speed scales well (execute only active parts)	–	+
Instruction-cache friendly (good locality)	+	–
Pipeline friendly (little/no branching)	+	–
WCRT predictable (simple control flow)	+	+/–
Low execution time jitter (simple/fixed flow)	+	–

[von Hanxleden, Duderstadt, Motika, et al.,
SCCharts: Sequentially Constructive Statecharts for Safety-Critical Applications,
PLDI'14]

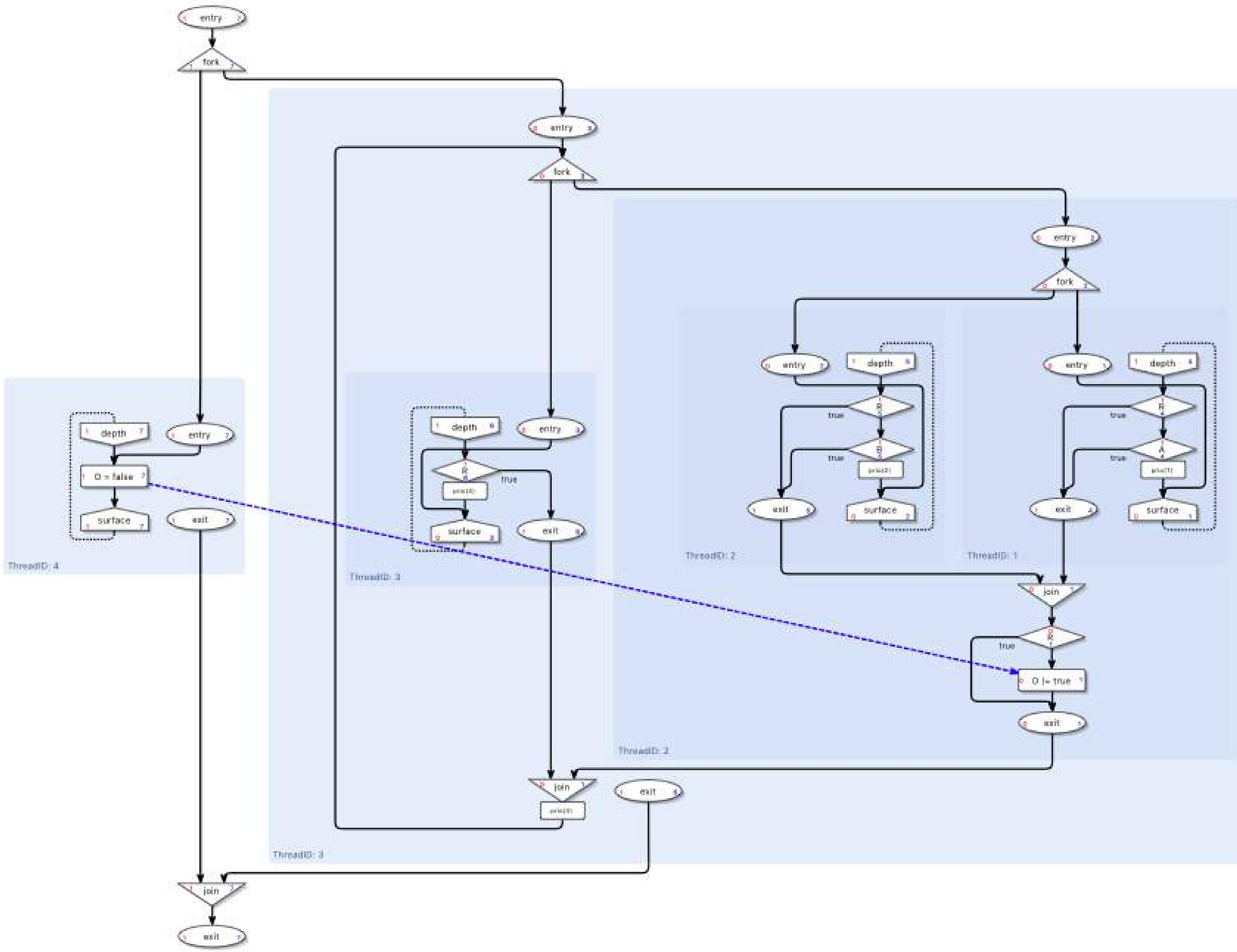
Compilation Option 1: Dataflow



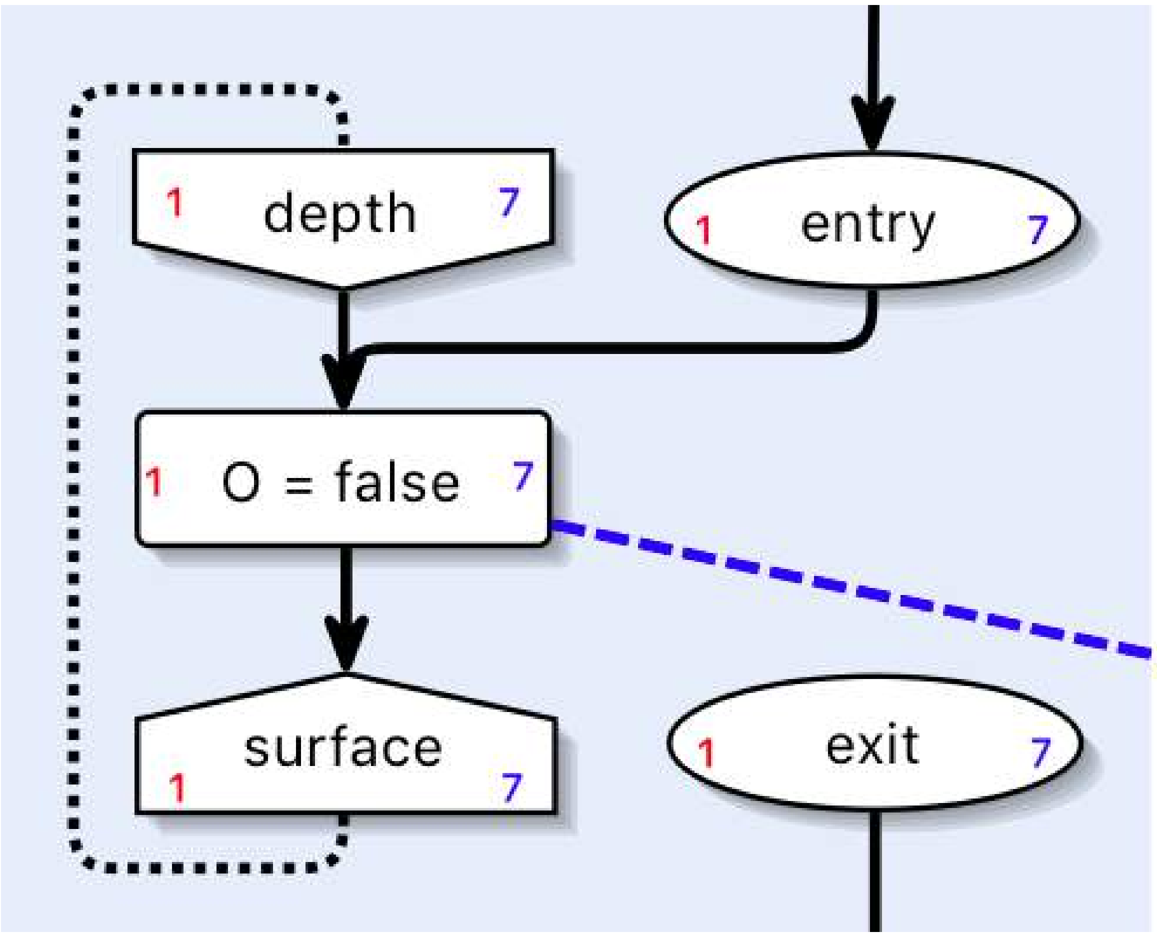
Dataflow SCG

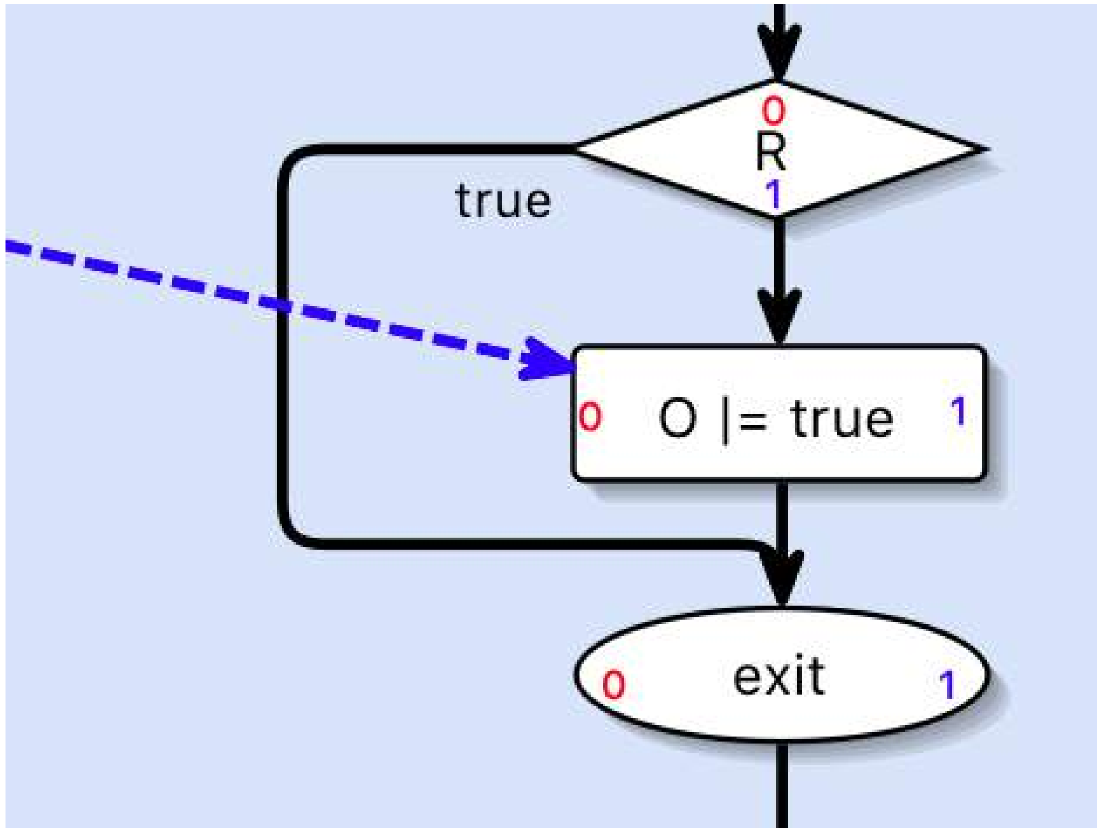


Compilation Option 2: Priority-Based



Compilation Option 2: Priority-Based

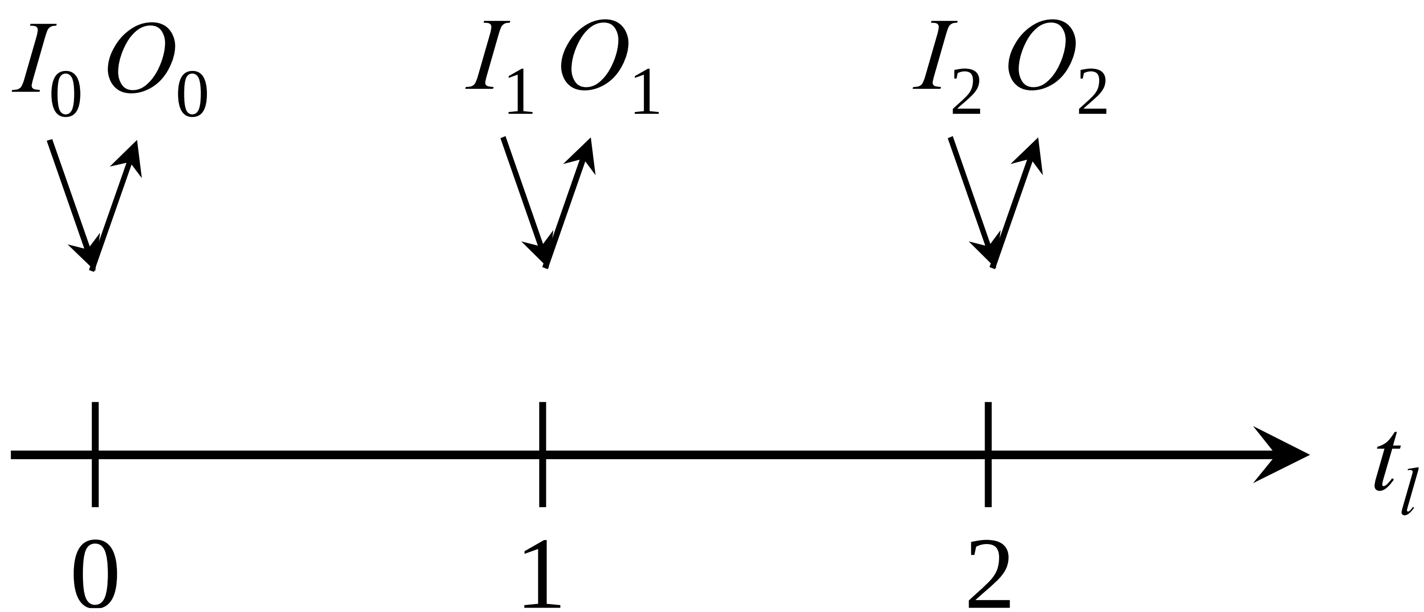




Priority-Based C Code

```
int tick() {

  tickstart(7);
  fork1(_region_0,_region_1, 3) {
    _region_0:
    O = 0;
    pause;
    goto _region_0;


  } par {
    _region_1:
    fork1(_region_2,_region_3, 2) {
      _region_2:
      prio(6);
      pause;
      if(R){

      } else {
        prio(3);
        goto _region_2;
      }


    } par {
    _region_3:
    fork1(HandleB,HandleA, 1) {
      HandleB:
      prio(5);
      pause;
      if(R){

      } else {
        if(B){

        } else {
          prio(2);
          goto HandleB;
        }
      }


    } par {
      HandleA:
      prio(4);
      pause;
      if(R){

      } else {
        if(A){

        } else {
          prio(1);
          goto HandleA;
        }
      }

    } join2(2, 5);
    if(R){

    } else {
      O |= 1;

    }

    } join2(3, 6);
    prio(3);
    goto _region_1;

  } join1(7);

  tickreturn();
}
```

SCCharts and Time



- Synchrony Hypothesis:
  Outputs are synchronous with inputs
- Computation "does not take time"
- Actual computation time does not influence result
- Sequence of outputs **determined** by inputs

Synchronous Execution

```
Initialize Memory
for each input event do
    Compute Outputs
    Update Memory
end
```

```
Initialize Memory
for each clock tick do
    Read Inputs
    Compute Outputs
    Update Memory
end
```

Fig. 1 Two common synchronous execution schemes: event driven (left) and sample driven (right).

[Benveniste et al., *The Synchronous Languages Twelve Years Later*, Proc. IEEE, 2003]

Multiform Notion of Time

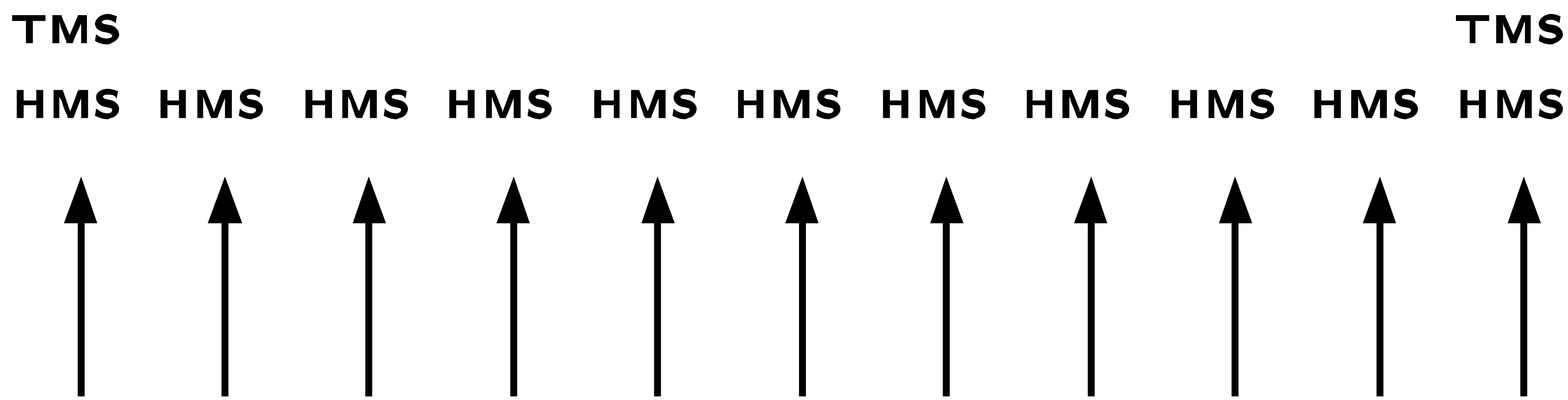
Only the simultaneity and precedence of events are considered.

This means that the physical time does not play any special role.

This is called multiform notion of time.

[https://en.wikipedia.org/wiki/Esterel]

Packaging Physical Time as Events



[Timothy Bourke, SYNCHRON 2009]

Event "HMS": 100 μsec have passed since last HMS

Event "TMS": 1000 μsec have passed since last TMS

A Problem With That ...

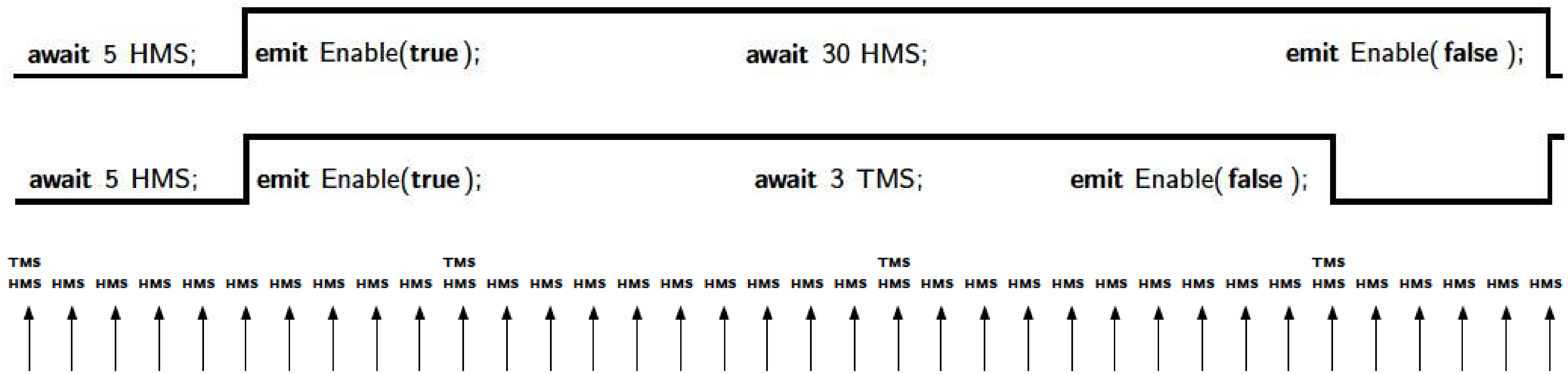
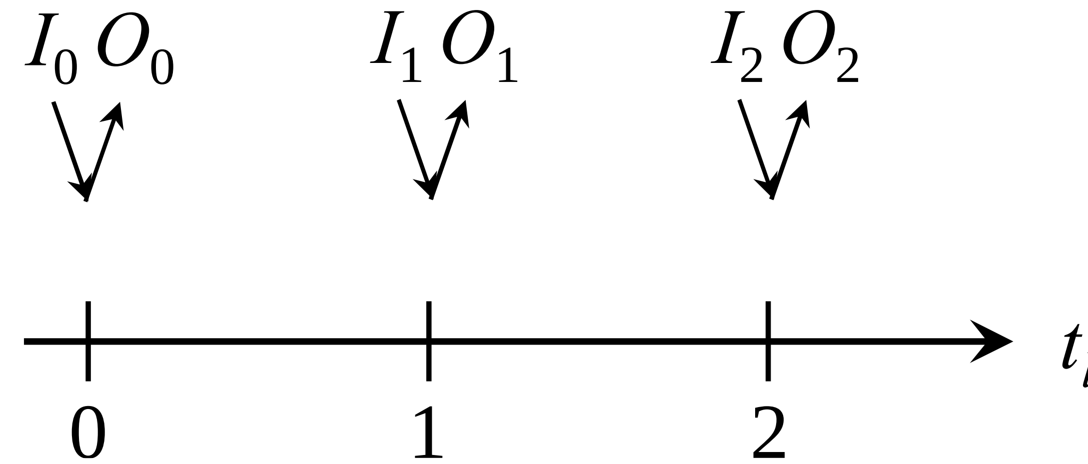


Fig. 4: Granularity of timing inputs

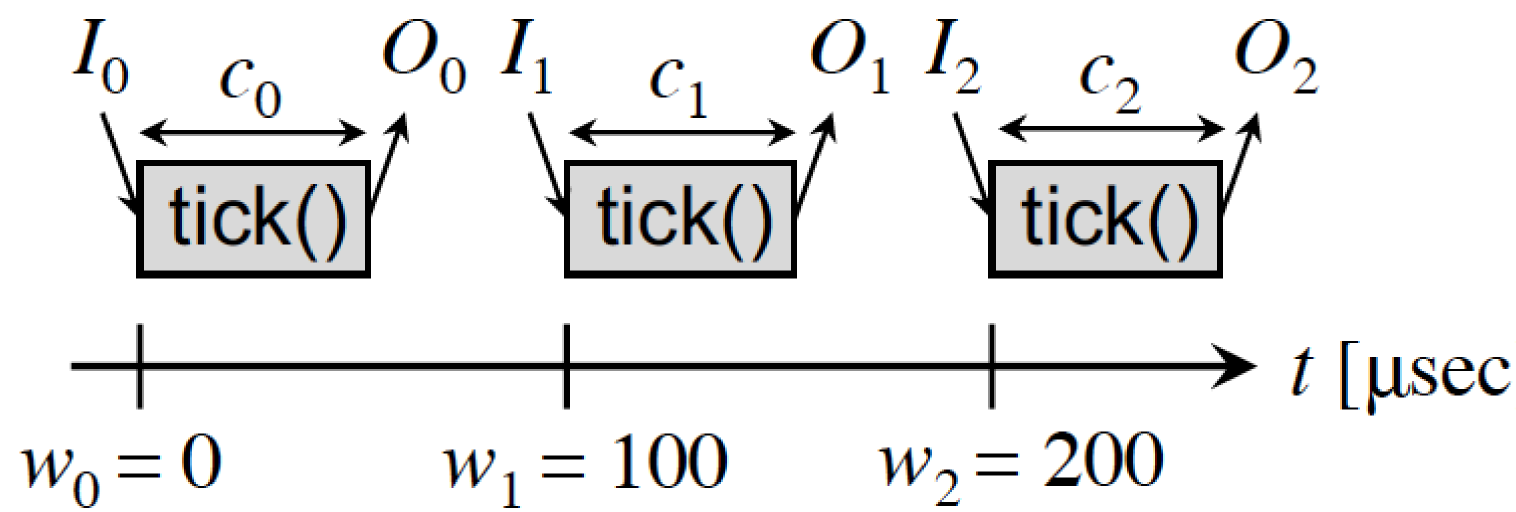
[Timothy Bourke, SYNCHRON 2009]

Dynamic Ticks

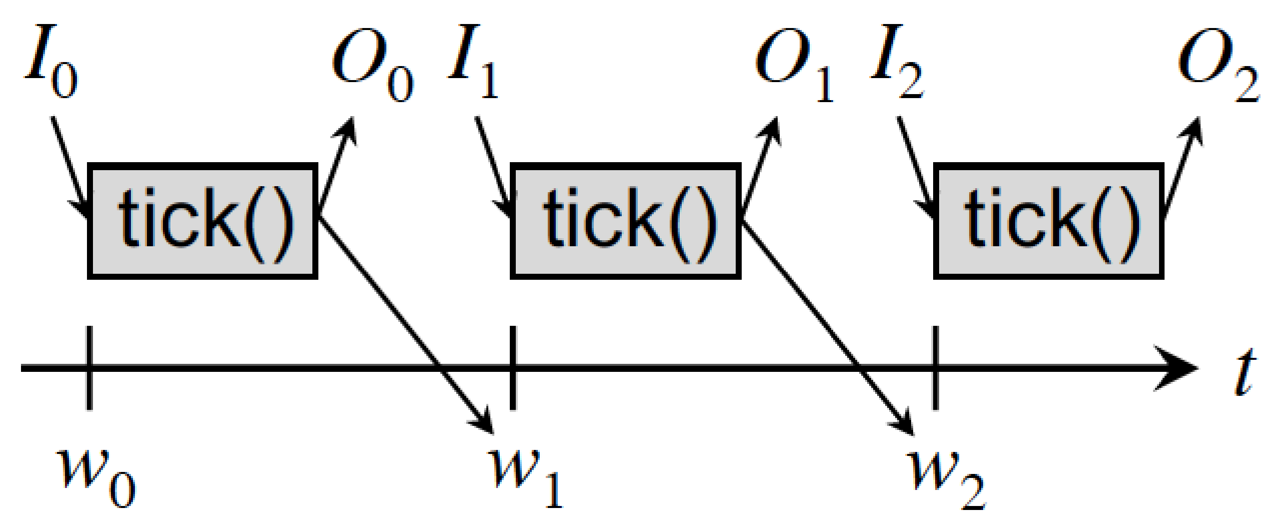
- Recall logical time:



- Physical time, time-triggered:



- Physical time, dynamic ticks:



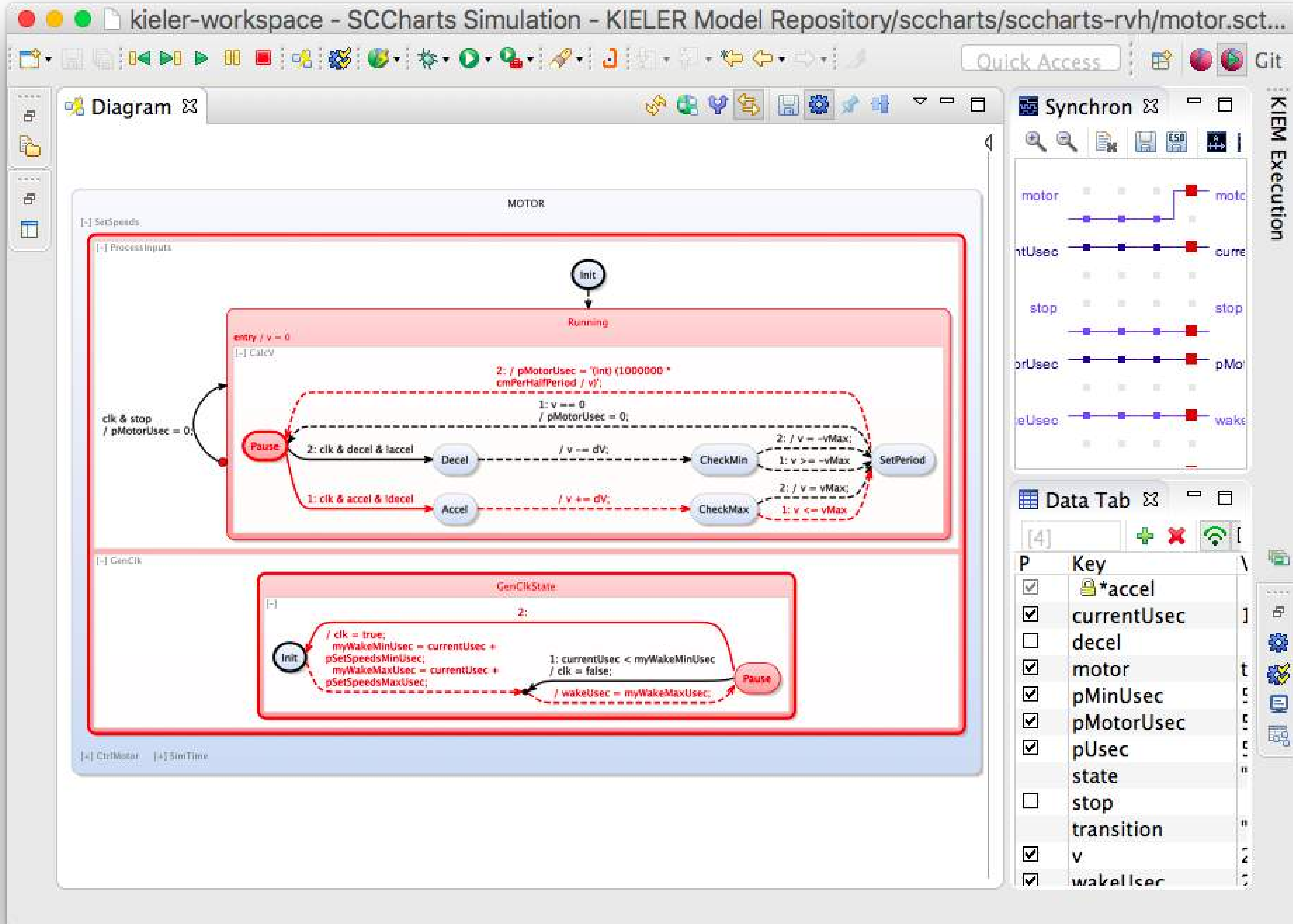
[von Hanxleden, Bourke, Girault,
Real-Time Ticks for Synchronous Programming, FDL'17]

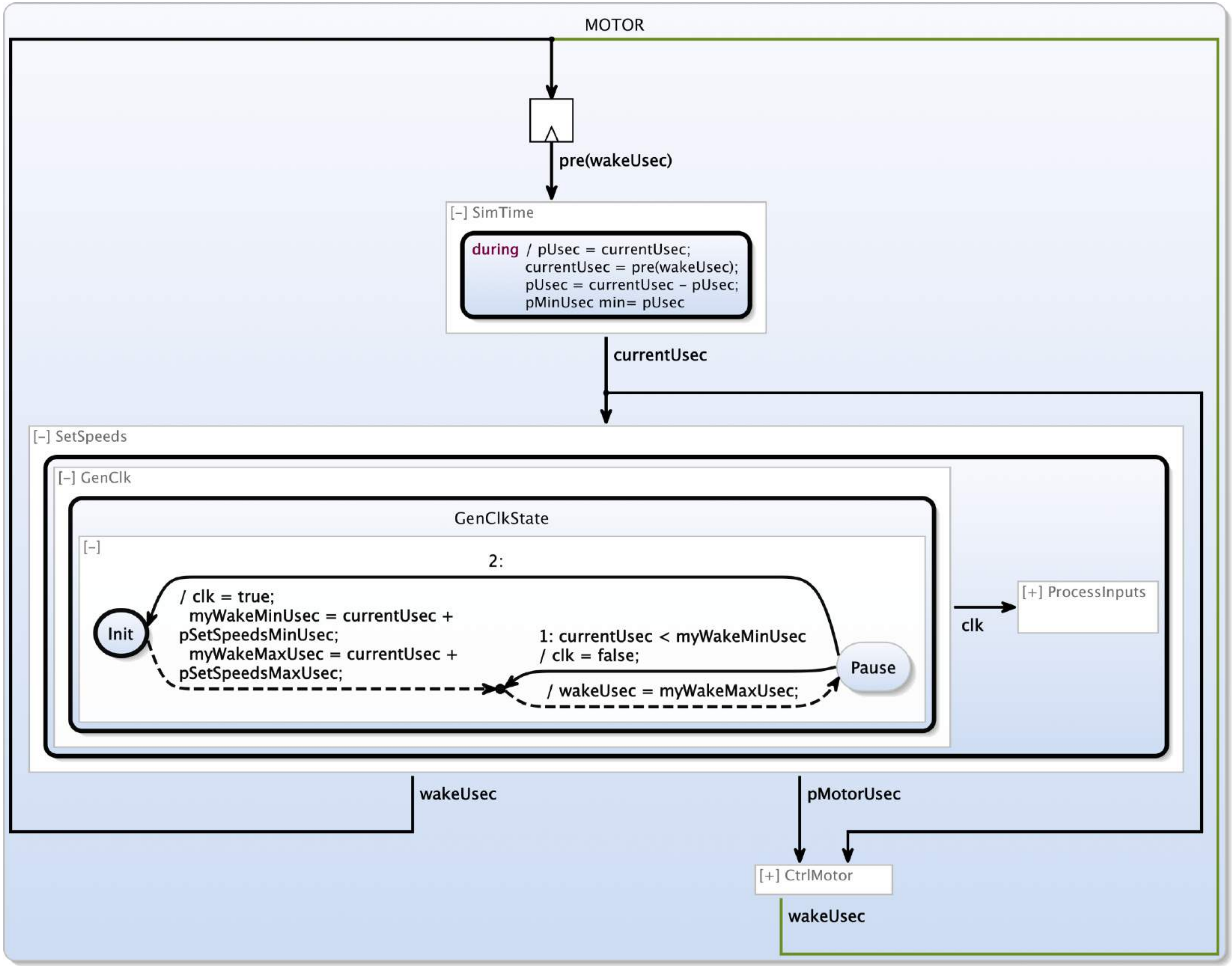
SCCharts Textual Representation

```
/** Controller for stepper motor
 */

scchart MOTOR {
  output int currentUsec = 0;  // [usec] Current simulated time;
                               // when deployed, this should be input
  output int wakeUsec;         // [usec] Time for next wake-up

  input bool accel, decel;     // Increase/decrease speed
  input bool stop;             // Emergency stop - sets (angular) speeds to 0

  output bool motor = false;   // Motor pulse
  output float v;              // [cm/sec] Robot speed
  output int pMotorUsec;       // [usec] Half period for motor

  int pSetSpeedsMaxUsec = 500000; // [usec] Maximum period of speed control loop
  int pSetSpeedsMinUsec = 400000; // [usec] Minimum period of speed control loop
  output int pUsec;            // [usec] Previous period (delta of wake-up times)
  output int pMinUsec = pSetSpeedsMaxUsec; // [usec] Minimum period

  float dV = 2;                // [cm/sec] Delta v applied during one speed
                                          // control loop cycle
  float vMax = 20;             // [cm/sec] Max speed of left/right motor
  float cmPerHalfPeriod = 1;   // [cm] Distance traveled by motor per half period
                                          // (duration of true or false)
```

Now use KIELER to synthesize graphical SCChart with ELK and simulate ...

```
/** Controller for stepper motor
 */

scchart MOTOR {
 output int currentUsec = 0;  // [usec] Current simulated time; when deployed, this should be input
 output int wakeUsec;        // [usec] Time for next wake-up

 input bool accel, decel;     // Increase/decrease speed
 input bool stop;            // Emergency stop - sets (angular) speeds to 0

 output bool motor = false;   // Motor pulse
 output float v;            // [cm/sec] Robot speed
 output int pMotorUsec;          // [usec] Half period for motor

 int pSetSpeedsMaxUsec = 500000; // [usec] Maximum period of speed control loop
 int pSetSpeedsMinUsec = 400000; // [usec] Minimum period of speed control loop
 output int pUsec;           // [usec] Previous period (delta of wake-up times)
 output int pMinUsec = pSetSpeedsMaxUsec; // [usec] Minimum period

 float dV = 2;             // [cm/sec] Delta v applied during one speed control loop cycle
 float vMax = 20;           // [cm/sec] Max speed of left/right motor
 float cmPerHalfPeriod = 1;   // [cm] Distance traveled by motor per half period (duration of true or false)

 // ================================================================================
 region SetSpeeds:

 initial state SetSpeeds "" {
  // Possible syntax, to replace the clk flag and the GenClk region,
  // and not having to condition delayed triggers with "clk &":
  // clock type = soft, min = pSetSpeedsMinUsec, max = pSetSpeedsMaxUsec
  bool clk; // Local clock

  // ================================================================================
  region ProcessInputs:

  initial state Init
  --> Running immediate;

  state Running {
   entry / v = 0;

   // ================================================================================
   region CalcV:

   initial state Pause
   --> Accel with clk & accel & !decel
   --> Decel with clk & decel & !accel;

   state Accel
   --> CheckMax immediate with / v += dV;

   state Decel
   --> CheckMin immediate with / v -= dV;

   state CheckMax
   --> SetPeriod immediate with v <= vMax
   --> SetPeriod immediate with / v = vMax;

   state CheckMin
   --> SetPeriod immediate with v >= - vMax
   --> SetPeriod immediate with / v = - vMax;

   state SetPeriod
   --> Pause immediate with v == 0 / pMotorUsec = 0
   --> Pause immediate with / pMotorUsec =
   '(int) (1000000 * cmPerHalfPeriod / v)';
  }
  o-> Running with clk & stop / pMotorUsec = 0;

// ================================================================================
  region GenClk:
  initial state GenClkState {
   int myWakeMinUsec, myWakeMaxUsec;


   initial state Init
   --> AssertWakeTime immediate with / clk = true;
   myWakeMinUsec = currentUsec + pSetSpeedsMinUsec;
   myWakeMaxUsec = currentUsec + pSetSpeedsMaxUsec;

   connector state AssertWakeTime
   --> Pause immediate with / wakeUsec = myWakeMaxUsec; // Initialize wakeUsec

   @layout[layerConstraint] LAST
   state Pause
   --> AssertWakeTime with currentUsec < myWakeMinUsec / clk = false
   --> Init;
  };
 };

 // ================================================================================
 region CtrlMotor:

 initial state CtrlMotor "" {
  // Possible syntax, to replace the clk flag and the GenClk region,
  // and not having to condition delayed triggers with "clk &":
  // clock type = hard, min = 1000000 * cmPerHalfPeriod / vMax  // = 50000
  bool clk; // Local clock

  // ================================================================================
  region GenClk:

  initial state GenClkState "" {
   int myWakeUsec;

   initial state Stopped
   --> AssertWakeTime immediate with pMotorUsec > 0 / myWakeUsec = currentUsec +
   pMotorUsec; clk = true;

   connector state AssertWakeTime
   --> Running immediate with / wakeUsec min= myWakeUsec; // Initialize wakeUsec

              @layout[layerConstraint] LAST
   state Running
   --> ResetClock with / clk = false;

   connector state ResetClock
   --> AssertWakeTime immediate with pMotorUsec > 0 & currentUsec < myWakeUsec
   --> Stopped immediate;
  };

  // ================================================================================
  region Motor:

  initial state Low
  --> High with clk / motor = true;

  state High
  --> Low with clk / motor = false;
 };

 // ================================================================================
 region SimTime:
 initial state SimTimeState "" {
  during / pUsec = currentUsec;
   currentUsec = pre(wakeUsec);
   pUsec = currentUsec - pUsec;
   pMinUsec min= pUsec;
  };
 }
```


kieler-workspace - SCCharts Simulation - KIELER Model Repository/sccharts/sccharts-rvh/motor.sct...
Quick Access
Git
Diagram
MOTOR
[-] SetSpeeds
[-] ProcessInputs
Init
Running
entry / v = 0
[-] CalcV
2: / pMotorUsec = '(int) (1000000 * cmPerHalfPeriod / v)';
1: v == 0
/ pMotorUsec = 0;
clk & stop
/ pMotorUsec = 0;
Pause
2: clk & decel & !accel
Decel
/ v -= dV;
CheckMin
2: / v = -vMax;
1: v >= -vMax
SetPeriod
1: clk & accel & !decel
Accel
/ v += dV;
CheckMax
2: / v = vMax;
1: v <= vMax
[-] GenClk
GenClkState
2:
/ clk = true;
myWakeMinUsec = currentUsec +
pSetSpeedsMinUsec;
myWakeMaxUsec = currentUsec +
pSetSpeedsMaxUsec;
Init
1: currentUsec < myWakeMinUsec
/ clk = false;
Pause
/ wakeUsec = myWakeMaxUsec;
[+] CtrlMotor
[+] SimTime
Synchron
motor
stop
Data Tab
[4]
P
Key
*accel
currentUsec
decel
motor
pMinUsec
pMotorUsec
pUsec
state
stop
transition
v
KIEM Execution



[Wechselberg, Schulz-Rosengarten, Smyth, von Hanxleden
Augmenting State Models with Data Flow
Principles of Modeling – LNCS Festschrift on Edward Lee's 60th Birthday (to appear)]